STREAMING ALGORITHMS FOR MATRIX APPROXIMATION

by

Amey Desai

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

in

Computing

School of Computing

The University of Utah

December 2014

The University of Utah Graduate School

STATEMENT OF THESIS APPROVAL

The thesis of **Amey Desai**

has been approved by the following supervisory committee members:

Jeffrey Phillips	, Chair	**10/13/2014**
		Date Approved
Feifei Li	, Member	**07/29/2014**
		Date Approved
Thomas Fletcher	, Member	**07/29/2014**
		Date Approved

and by **Ross Whitaker**, Chair/Dean of the Department/College/School of **Computing**

and by David B. Kieda, Dean of The Graduate School.

ABSTRACT

Matrices are ubiquitous in data analysis. Most real-world datasets are formulated as $n \times d$ matrix A, where n represents the number of data points and d represents the number of features. In addition, matrices are also used to store pairwise similarities between data points. Performing any large-scale machine-learning task in an efficient manner needs large matrices to be stored in memory, while also having them distributed across various machines. Since most datasets have a lower intrinsic dimension for the actual signal, a smaller sketch matrix approximates the original matrix in addition to giving a low-rank matrix, which reduces the memory requirements of machine-learning tasks. Computing low-rank matrices for best approximation accuracy is done via Singular Value Decompostion (SVD), which is computationally expensive and is not suitable in distributed environments. In this thesis, we survey various algorithms for matrix approximation and present improvements over existing work.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First off, it has been a great pleasure to work with Jeff Phillips. I started with little knowledge of the problem, but Jeff has helped me in understanding the smallest details. Jeff has helped me debug some of the randomized algorithms that go beyond the standard realms of software engineering. He has also improved my communication skills and leaves me with cleaner and clearer ideas every time I communicate with him.

I'm grateful to Feifei and Tom for agreeing to be on my committee and being very flexible and supportive throughout my work. I would also like to thank Mina for all her help and support. Robert Ricci and Leigh Stoller helped us in providing the testbed APT for performing experiments and providing customizations for various scenarios. Without them the experimental scale would have been much smaller and I'm very fortunate they worked with me. I'm also thankful to Valerio Pascucci, Yarden Livnat, and Peer-Timo Bremer for letting me work with them on various data analysis and visualization projects.

Ann has always been super supportive and worked with me through formalities of the thesis, and I appreciate her moral support throughout my 2 years at Utah. In my past year at SCI, Brenda and Tony have always been there to help and cheer me up, and the support staff at the whole school is incredible.

My friends Abhishek and Naveen have always helped me with the smallest details regarding thesis formalities. Jotham, Abhishek, Mike, and Ankit have been great friends. I'm glad to have known them in the past 2 years.

Lastly I would like to thank my awesome mother for all her love and support.

CHAPTER 1

INTRODUCTION

Matrix approximation is an important tool in data mining [6]. Prominent applications of matrix approximation by Singular Value Decomposition (SVD) include recommendation systems [18], information retrieval via Latent Semantic Indexing [7, 45], Kleinberg's celebrated Hyper Induced Text Search (HITS) algorithm for web search [35], clustering [15, 39], and learning mixtures of distributions [4, 32], to name just a few. In many applications, datasets are naturally formulated as an $n \times d$ matrix A in which n objects are described by d features. Examples of such objects include documents, images, and stocks. Examples of corresponding features are terms, temporal resolution, environmental conditions. It is well known that SVD gives the best approximation to a matrix under any unitarily invariant norm. Although exact SVD methods are polynomial, they are computationally intensive when performed exactly. For example, dense SVD methods require $O(nd^2)$ time and $O(nd)$ on a $n \times d$, $(d \leq n)$ matrix [27], both of which are prohibitively large for moderate-size datasets.

Today's datasets are generated at a rapid rate, and the size of datasets exceeds the internal memory requirements. Hence efficient algorithms are needed for accessing data over external storage. Multiple passes over such datasets are also prohibitive, especially for datasets generated over the web. Hence, we restrict ourselves to the pass efficient "streaming" model of computation [47]. Here access to the input is limited to a constant number of sequential scans, and RAM usage depends sublinearly on input size. Thus, we look at computing a sketch matrix as an approximation of the original matrix that can adhere to the memory requirements. The sketch matrix is close enough to the original matrix and queries which are performed on the original matrix, can instead be answered by the much smaller sketch matrix with a small loss in accuracy.

Computing SVD in a streaming setting is difficult due to its time and space requirements. Even for sparse datasets, iterative SVD methods [27] alone are not suitable for streaming models as their convergence speed is unknown a priori and requires multiple passes over the datasets.

1.1 Applications

We look at some real-world applications of low-rank approximation to see the feasibility of the given problem.

1.1.1 Image Compression

Datasets usually have a noise component. For example, when we take photograph with cameras, various sensors inside the cameras introduce a small noise component, which when taking pictures becomes encoded in the picture. In Figure1.1 (b), we observe that after 100 singular values, the remaining singular values are very close to 0 and are relatively small. Most of the singular values after the first 100 singular values correspond to noise instead of any features of the image. Noise leads to a higher-rank matrix than desired, and computing a low-rank matrix approximates the original matrix while removing the noise. A low-rank matrix also reduces the space needed to store the low-rank image and has the added advantage of working with a smaller dataset, leading to better running times for any image processing operations. In Figure 1.2, we see low-rank images obtained via SVD for rank $r = (20, 50, 100, 150)$. Each of those images will take less space than the original image. The dimensions for the original image are 512×512, which is 262144 numbers/entries of a matrix. By taking a low-rank approximation using the SVD, we get three matrices, U, S and V_t. For rank 20, the dimensions of U are 512×20, S is 20, and V_t is 20×512. Adding the numbers for the given three matrices yields 20500 numbers or entries, which is about 7.82% of the original number size. Also a matrix with a rank at least 100 is a good approximation of the original image.

1.1.2 Latent Semantic Indexing

Latent Semantic Indexing (LSI) [45] is an information retrieval technique for identifying patterns between words and context. It is formulated as a matrix A where there are n documents and d words making up the $n \times d$ data matrix. Each entry of the matrix counts

the number of occurrences of a given word in a specific document. The count can have a weighted component based on domain-specific knowledge. It is easy to see that the core topics in this given matrix will be much smaller, and hence the matrix should have a smaller rank, which can be computed via the SVD.

1.1.3 Principal Component Analysis (PCA)

One of the steps of PCA is to find the sample covariance matrix K by multiplying the data matrix with itself AA^T. By computing small sketch matrices B, where B matrix is of much smaller rank than the original matrix, we can approximate AA^T by BB^T. Here the matrix B is a $k \times d$ where $k \ll d \ll n$. The sketch matrix is particularly useful as the matrix multiplication for BB^T will be much faster than the original matrix and the running time for PCA is improved significantly.

1.2 Motivation

As we see in the applications, computing a low-rank approximation is useful in practice, but using SVD has limitations, especially in a streaming setting. To circumvent this problem, multiple algorithms have been proposed.

Matrix approximation has received attention from the theoretical computer science community. Various algorithms to compute matrix approximation have been proposed [10, 17, 19, 21, 22, 26, 37, 49], but a comprehensive empirical analysis of this area has been lacking, because not all algorithms have a common framework to measure against each other.

Many algorithms give theoretical bounds with different notions of projection error, e.g., Leverage Score Sampling uses pseudo inverse as a projection operator and also a rank reducing procedure [19, 46]. Clarkson and Woodruff introduced two one-pass projection error computations, which make comparisons more difficult with other algorithms. In addition, some of the algorithms need $k - wise$ independent hashing, which is hard to do in practice. For a given problem such as image compression, there is not a clear answer as to which algorithms might be a good fit for the approximation accuracy given acceptable running times and space requirements.

Also, various algorithms give a relative error approximation, but an important question to answer is whether this algorithm actually works well in practice. This question needs to

be answered at two levels; one being the running time and the other being how good the approximation is via covariance error and projection error. We answer both questions in this manuscript.

Also, computing the SVD in a streaming setting does not give the flexibility for any kind of online queries. SVD needs the whole dataset to be present in memory and any intermediate results cannot be computed. Hence streaming algorithms for computing matrix approximation are more appealing.

1.3 Thesis Statement

The formal description of the problem is as follows:

Problem 1.3.1. *Given a matrix $A \in R^{n \times d}$ with rank ρ, we want to find a matrix $\hat{A}$ with lower rank $k \ll \rho$ such that $\|A - \hat{A}\|_\zeta$ is close to $\|A - A_k\|_\zeta$ for $\zeta = 2, F$, where A_k is the best rank k approximation of A.*

A low-rank matrix is computed by first computing a sketch matrix B that approximates A and then by projecting the matrix A on an orthogonal subspace spanned by sketch matrix B.

Three algorithmic areas for matrix approximation are implemented and studied:

- Column/Row Sampling
- Random Projections
- Frequent Directions

1.4 Contributions

The main contributions of this thesis are :

- We showcase some existing techniques outside of low-rank approximation literature that perform empirically better or as well as the existing algorithms in both accuracy and time. The techniques are:
  - Weighted Reservoir Sampling
  - Priority Sampling

  - VarOpt Sampling
  - Subspace embedding using Toeplitz matrix
  - Freqent Direction variants and improving running time of Frequent Directions variants
  - Subspace embedding using exponential random variables

- In particular, new techniques we develop, PRIORITY SAMPLING and Tweak - PARAMETERIZED FREQUENT DIRECTIONS, provide best in class with respect to certain criteria.

- We provide a thorough comparison of existing algorithms as well as the algorithms mentioned above.

We compare algorithms for the following generic parameters:

- How well does the sketch matrix product $B^T B$ approximate $A^T A$?

- How good is the low-rank approximation obtained via sketch matrix B ?

- How fast can sketch matrices be computed?

- How much space do the sketch matrices take?

We focus on answering the following specific questions:

- What algorithms give the best approximation accuracy with the smallest possible space?

- What algorithms give the best running time and approximation accuracy with no space constraints?

1.5 Organization

Chapters 2 and 3 describe the necessary linear algebra background. Chapter 4 gives details about the algorithms implemented. Chapters 5 and 6 describe the experimental setup and the results obtained, respectively.

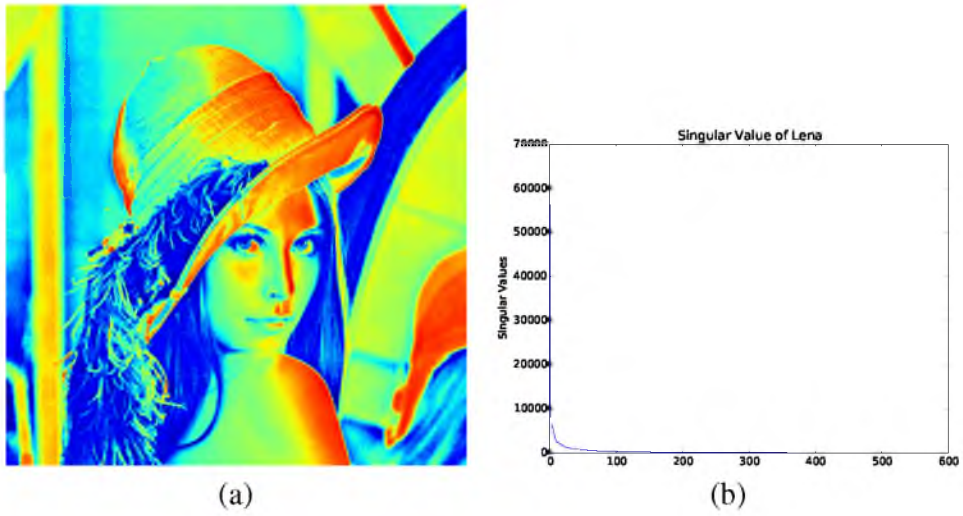

Figure 1.1. Lena and its singular value distribution, a) Original Lena image, b) Singular value distribution of Lena image

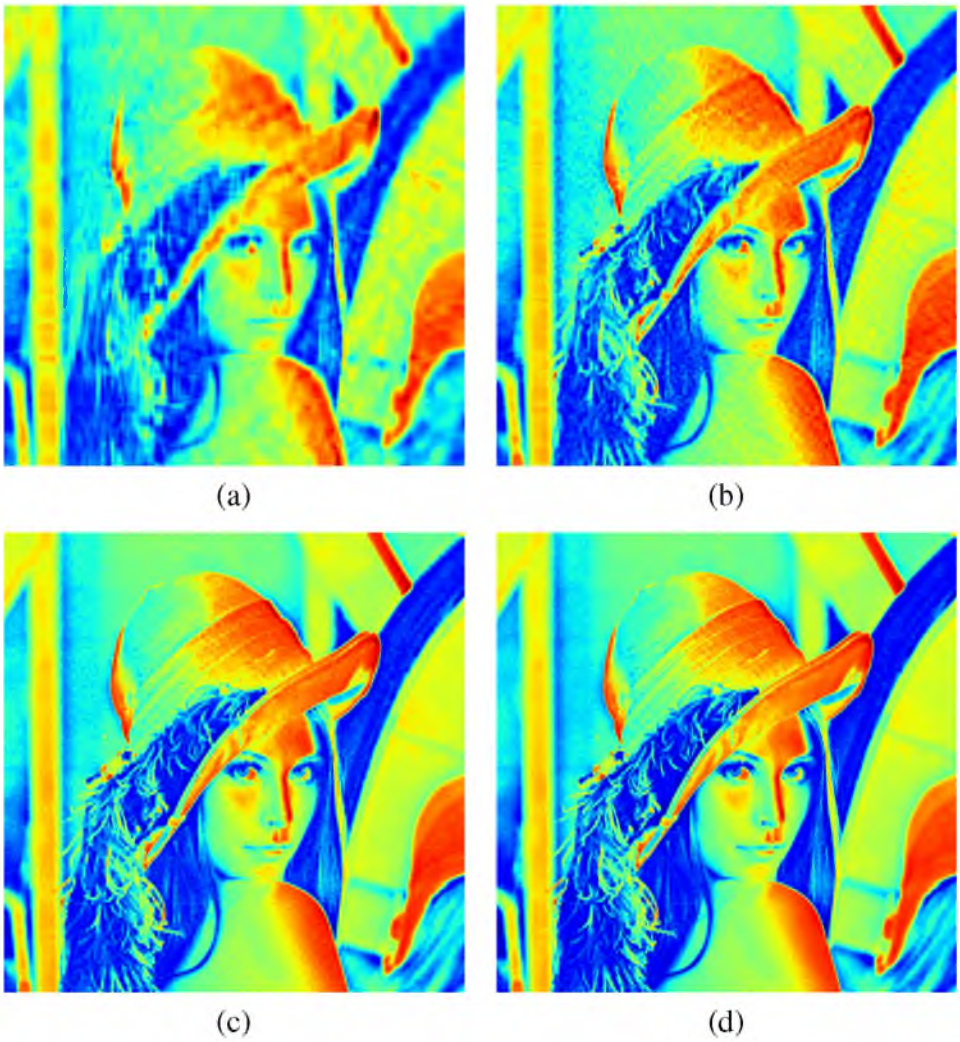

Figure 1.2. Lena low-rank images, a) Rank-20 approximation image, b) Rank-50 approximation image, c) Rank-100 approximation image, d) Rank -150 approximation image

CHAPTER 2

PRELIMINARIES, NOTATION, AND MOTIVATION

Throughout this manuscript, we assume input data are in the form of a matrix $A \in \mathbb{R}^{n \times d}$ where each row a_i (of length d) is one data point.

2.1 Linear Algebra Review

For a vector $x \in R^n$, we let $\|x\| = (\sum_{i=1}^{n} x_i^2)^{1/2}$ denote the Euclidean length. For a matrix $A \in \mathbb{R}^{n \times d}$, we let a^j for $j = 1, \ldots, d$ denote the j-th column of A and a_i for $i = 1, \ldots, n$ denote the i-th column of A.

If the inner product of two vectors is zero, then the two vectors are said to be orthogonal. If the orthogonal vectors are of unit length, then they are said to be orthonormal to each other. A matrix is said to be orthogonal if $AA^T = I$.

We denote matrix norms by $\|A\|_\xi$ where $\xi = F, 2$.

The Frobenius norm of matrix A is defined as $\|A\|_F^2 = \sum_{i=1}^{n} \|a_i\|^2$, and the Spectral norm is defined as $\|A\|_2 = sup_{x \in \mathbb{R}^n, x \neq 0} \frac{\|Ax\|}{\|x\|}$. These norms are related to each other as $\|A\|_2 \leq \|A\|_F$.

Matrix norms are used to see how two matrices differ. If we compute a low-rank matrix, then we look at the difference between the original matrix and the approximated matrix in the Frobenius norm, to see how these two matrices differ from each other. The difference between two matrices over the Frobenius norm measures the root-mean square error of the matrix, which shows the average effect of noise in the two matrices. The Spectral norm is the largest singular value of a given input matrix and hence takes more computation time. We use the Frobenius norm error as it is easier to compute.

The rank of a matrix A is the number of linearly independent columns (rows) of matrix A and is denoted by $\mathsf{rank}(A)$. For every matrix the column rank is equal to the row rank.

Given a row r and a matrix B, let $\pi_B(r)$ be a *projection* operation of r onto the subspace spanned by B. In particular, we will project onto the row space of B, which can be written as $\pi_B(r) = rB^T(BB^T)^+B$ where B^+ indicates taking the Moore-Penrose psuedo-inverse of B.

We denote nnz(A) as the number of nonzero entries present in a given matrix A.

2.2 Best Rank-k Approximation

We already stated the low-rank approximation problem in Chapter 1. Here, we look at the problem geometrically. Given a matrix $A \in \mathbb{R}^{n\times d}$ with rank ρ, we look to find a matrix of rank k, where $k \ll \rho$, which minimizes the Frobenius norm of their difference. Given data points as $a_1, a_2, a_3, ..., a_n \in \mathbb{R}^d$, we look to find a $k-$ dimensional linear subspace Q such that the sum of the squared distances of the points to the subspace is minimized, which can be expressed as follows :

$$(\sum_{i=1}^{n} dist(a_i, Q)^2)^{1/2}$$

The SVD method decomposes A into three matrices, $U \in \mathbb{R}^{n\times n}$, $S \in \mathbb{R}^{n\times d}$, and $V \in \mathbb{R}^{d\times d}$, such that $A = USV^T$.

Matrices U and V are orthonormal in columns, and are called left singular vectors and right singular vectors, respectively. The matrix S is a diagonal matrix where each entry on the diagonal is nonzero and sorted in descending order. The nonzero entries are called singular values. $\{\sigma_1, \ldots, \sigma_d\}$.

We refer to the best rank-k approximation of A as $A_k = U_k S_k V_k^T$, where U_k, S_k, and V_k are the first k columns of each matrix.

Computing the SVD takes $O(nd\min(n, d))$ time and storing A_k requires $O(nd)$ space. One can reduce the space usage to $O((n + d)k)$ by storing the set of matrices $\{U_k, S_k, V_k\}$. Moreover, even the set $\{U, S, V\}$ takes only $O(nd + d^2)$ space since we can drop the last $n - d$ columns of U, and the last $n - d$ rows of S without changing the result.

2.3 Error Bounds

Denoting the approximated matrix as $\hat{A}$, the approximation error is bounded as $\|\hat{A} - A_k\|_\xi)$ by some factor of $\|A - A_k\|_\xi$ and/or $\|A\|_\xi$. Based on the type of bounds the algorithms provides, they fall into one of the caterogies listed in the following section.

2.3.1 Construction Bounds

This group of algorithms constructs $\hat{A}$ as a multiplication of a few smaller matrices that are stored. Note that matrix $\hat{A}$ might not get constructed and stored explicitly. The well-known CUR decomposition [20] is an example.

2.3.2 Projection Bounds

These algorithms find a suitable subspace Q for rowspace or columnspace of A and constructs $\hat{A}$ as the projection of A onto this subspace, i.e., $\hat{A} = \pi_Q(A)$. There are two variants for it.

- Projection Error 1: Projection onto k subspace and then get approximation

  $\mathsf{proj\text{-}err1} = \|A - \pi_{Q_k}(A)\|_F^2 / \|A - A_k\|_F^2$

- Projection Error 2: Projection onto subspace and then get k approximation

  $\mathsf{proj\text{-}err2} = \|A - (\pi_Q(A))_k\|_F^2 / \|A - A_k\|_F^2$

Here, we are computing the ratio of the approximated matrix and the best possbile approximation given by SVD. The error should be at most 1. An error of 1 would give the best approximation.

2.3.3 Covariance Error

Covariance error is defined as :

$$\mathsf{err} = \|A^T A - B^T B\|_2 / \|A\|_F^2$$

Here A is the input matrix and B is a small sketch matrix. In PCA, $A^T A$ is used for computing the covariance matrix, which can be replaced by $B^T B$ if the given error is very small. $A^T A - B^T B$ gives the difference between two vectors. Let the difference be C. The 2-norm of C gives the direction/feature along which the matrices $A^T A$ and $B^T B$ differ the most. We divide C by the Frobenius norm of the matrix because if the matrix $B^T B$ is zero, then the 2-norm of $A^T A$ will equal the Frobenius norm of A. Algorithms try to obtain an error value close to 0.

2.3.4 Additive Error

The ε-additive error bound is of the form $\|\hat{A} - A_k\|_\xi^2 \leq \|A - A_k\|_\xi^2 + \varepsilon\|A\|_\xi^2$, which is called an additive error bound due to its dependence on an additional term $\|A\|_\xi$. It is a weak error bound and in the case of large $\|A\|_{F,2}$ provides poor guarantees.

2.3.5 Relative Error

The ε-relative error bound is the stronger error bound dependent only on the "best rank-k error" term and is denoted as $\|\hat{A} - A_k\|_\xi^2 \leq (1 + \varepsilon)\|A - A_k\|_\xi^2$.

2.4 Streaming Model

2.4.1 Definition

In the streaming model, input items are presented in a sequence one after the other and ideally we look at items only in only one pass. We look at a row-wise streaming model, where an item is a row of a matrix. In this model we have limited memory, much less than the input matrix, and hence we look at producing a sketch of the original matrix. The idea is to generate this sketch in one pass and perform queries on the sketch instead of the original matrix.

2.4.2 Evaluation

Streaming algorithms are evaluated on three parameters in addition to the error:

- The number of passes, ideally one
- Memory used
- Running time of the algorithm

CHAPTER 3

ALGORITHMS

We cover three major algorithm areas for matrix approximation.

- Row/Column Sampling
- Random Projections
- Frequent Directions

3.1 Column/Row Sampling

The Column/Row Sampling technique selects a subset of columns/rows that best represent the original matrix. We present some algorithms [17, 21–23] targeted towards tall and skinny matrices. The general idea in this area is to select rows/columns, that are important and various algorithms define importance differently. This technique is also known as importance sampling.

File records or IP addresses have associated weights such as the size of the file or the number of times the IP address makes a request, respectively. To estimate the overall weight of the items using sampling, it is necessary that the sample estimate captures the heavy items, otherwise the variance would be high. Hence, the intuition for sampling algorithms is an item with more weight should have a high probability of being included in the sampling estimate and appropriately rescaled to preserve norms.

The work in [23] picks $\ell = 10^7 \max\{k^4/c^3\varepsilon^2, k^2/c^3\varepsilon^4\}$ rows of A independently at random, each according to a probability distribution satisfying $p_i \geq c\frac{\|A_i\|^2}{\|A\|_F^2}$. These rows get rescaled by a factor of $1/\sqrt{\ell * p_i}$ and form a $\ell \times d$ matrix B. Note that if $c = 1$, this scaling amounts to normalizing all rows to be of the same length. The way the rows are selected is what we interpret as importance sampling. The underlying intuition is to pick rows based

on the squared length of the row (Euclidean Norm), which amounts to choosing the rows, that contribute the most mass for the matrix.

It is shown in [23] that $B^TB \sim A^TA$. Since the (i,j)th entry of A^TA is the dot product of the ith and jth column of A, and B has a random sample rows of A, the entry $(B^TB)_{(i,j)}$ estimates the corresponding entry for A^TA and scaling makes this estimate unbiased. Since the right singular vectors of a matrix A are the eigenvectors of A^TA, the eigenvectors of B^TB are sufficient to approximate the right singular vectors for A.

The algorithm given by [23] achieves a projection error bound on the Frobenius norm: $\|A - \pi_{B_k}(A)\|_F^2 \leq \|A - A_k\|_F^2 + \frac{10k}{cs}\|A\|_F^2$. The algorithm takes time polynomial in k, $1/\varepsilon$ and $\log(1/\delta)$ and independent of n and d. The bounds are given only in terms of the Frobenius norm.

One of the problems associated with the above approach is the higher-order polynomial in k and ε which makes it impractical. This was the first line of work to introduce sampling to do a low-rank approximation to the best of our knowledge.

The generic algorithm for importance sampling is given in algorithm 3.1.1.

3.1.1 LinearTime SVD

Drineas *et al.* [17] modified the above-mentioned [23] algorithm such that both the construction and computation of the low-rank matrix is done in linear time in $\max(n,d)$ for LinearTime SVD. In addition, they improved polynomial dependence on k and provided both the Frobenius and Spectral norm bounds, respectively. They present two algorithms: "LinearTime SVD" and "ConstantTime SVD." We do not focus on "ConstantTime SVD" as the bounds associated are higher-order polynomials, which makes it undesirable. The algorithm for LinearTime SVD is given in algorithm 3.1.2.

Algorithm 3.1.1 (Generic) importance sampling steps

1: **Input:** $\ell, \varepsilon \in (0,1], A \in \mathbb{R}^{n\times d}$
2: $B_0 \leftarrow$ all zeros matrix $\in \mathbb{R}^{\ell\times d}$
3: **for** $i \in [n]$ **do**
4: Compute probability p_i for row i
5: **for** $j \in [\ell]$ **do**
6: Insert a_i into B by sampling with replacement
7: **return** B

Algorithm 3.1.2 Linear Time SVD

1: **Input:** $\ell, \epsilon \in (0,1], k, A \in \mathbb{R}^{n \times d}$
2: $C \leftarrow$ all zeros matrix $\in \mathbb{R}^{\ell \times d}$
3: **for** $i \in [n]$ **do**
4: Compute probability $p_i = |a_i|^2 / \sum_{j=1}^{i} |A_{(j)}|^2$ for row a_i
5: **for** $j \in [\ell]$ **do**
6: Insert a_i into C by sampling with replacement
7: Rescale each row as, $C_j = a_i / \sqrt{(l * p_i)}$
8: Compute $C^T C$ and its SVD, $C^T C = \sum_{t=1}^{\ell} \sigma_t^2(C) y^t y^{t^T}$
9: Compute $h_t = C y^t / \sigma_t(C)$ for $t = 1...k$
10: **return** B_k where $B_{k(t)} = h_t$

The LinearTimeSVD algorithm samples $c \geq 4k\eta^2/\beta\varepsilon^2$ rows to satisfy the Frobenius norm bound and needs to sample $c \geq 4\eta^2/\beta\varepsilon^2$ rows for the Spectral norm bound, where $\eta = 1 + \sqrt{(8/\beta)\log(1/\delta)}$ and β is a positive constant $\beta \leq 1$. The resulting matrix is $C \in R^{\ell \times d}$. Note that left singular vectors and values of C will be approximations to left singular values and vectors of A. In order to compute them in an efficient way, instead of taking the SVD of C directly, this method takes the SVD of $C^T C$. Getting its right singular vectors it computes left singular vectors of C, as given in steps 8 and 9 in algorithm 3.1.2.

Their additive error bounds are as follows :

$$\|A - B_k B_k^T A\|_{F,2}^2 \leq \|A - A_k\|_{F,2}^2 + \varepsilon \|A\|_F^2$$

where H_k is the rank k left singular matrix of C

Using the SELECT algorithm, LinearTime SVD can be done in one pass over the data. The SELECT algorithm in [16] lets us sample rows with probability $p_i = |A_{(i)}|^2 / \sum_{j=1}^{i} |A_{(j)}|^2$, which ensures that, at the end, each row is sampled with probability $|A_{(i)}|^2 / \|A\|_F^2$.

For computing the probability of each row to be selected, each element of the matrix needs to be accessed once, which needs $O(nnz(A))$. The running time to compute the sketch matrix C is $O(nnz(A))$. Computing the matrix product $C^T C$ needs $O(\ell^2 d)$ time, and its SVD takes (ℓ^3) time. Thus, the total running time for LinearTime SVD for computing a low-rank approximation is $O((\ell^2 d) + (\ell^3) + nnz(A))$. We focus on the running time for computing the sketch matrix going forward.

The sketch matrix needs $O(nnz(B))$ space, and for computing the probability for each row using SELECT algorithm, needs $O(\ell)$ space. Hence the total space needed is

$O(nnz(B) + \ell)$

3.1.2 Leverage Scores

Leverage scores are tied up with the problem of selecting the "best" columns from a data matrix. This problem is called the Column Subset Selection Problem (CSSP) and formally is defined as below:

Let $A \in \mathbb{R}^{n \times d}$ and let $c < d$ be a sampling parameter. Find a matrix $B \in \mathbb{R}^{n \times c}$ containing c columns of A, such that it minimizes the Frobenius norm $\|A - CC^{+}A\|_F$ or Spectral norm $\|A - CC^{+}A\|_2$, where C^{+} is the Moore-Penrose pseudoinverse of C.

Intuitively, leverage scores are statistics about matrix A that determine the most representative columns/rows, with respect to a rank-k approximation. The formal definition is as follows:

Let $V_k \in \mathbb{R}^{d \times k}$ contain the top k right singular vectors of the matrix $A \in \mathbb{R}^{n \times d}$. Then the rank-$k$ leverage scores of the i-th column of A are defined as

$$\ell_i^{(k)} = \|[V_k]_{i,:}\|^2$$

where $[V_k]_i$ denotes the i-th row of V_k. The history of leverage score sampling dates back to an old work by Joliffe [30], in which he proposed a deterministic approach for sampling columns of A corresponding to the largest leverage scores. Joliffe's algorithms have been shown to perform very well empirically [9, 31]. In [20], Drineas *et al.* came up with a straightforward extention of Joliffe's algorithm as a randomized algorithm, which allowed them to sample columns of A with a probability proportional to the relative weight of leverage scores, described in detail below. They showed $c = O(k \log k/\varepsilon^2)$ sample columns are needed to achieve a $(1 + \varepsilon)$-relative error bound.

In [8], Boutsidis *et al.* came up with two deterministic and two randomized algorithms based on a well-known lemma about matrix factorization. This lemma is mentioned in section 3.1 in their paper. Their best result samples $c = \frac{2k}{\varepsilon}(1 + o(1))$ columns and achieves a relative error Frobenius norm bound in expectation. In their best result, they use adaptive sampling [14] as a subroutine.

We describe the algorithm given by Drineas *et al.* in [19, 20] as SUBSPACE SAMPLING.

3.1.3 Subspace Sampling

The best rank-k approximation obtained via the truncated SVD has eigenrows/eigencolumns that are linear combinations of the original data, which makes interpreting them difficult in downstream analysis. SUBSPACE SAMPLING is motivated where the best rank-k approximation matrix is expressed via a smaller and important set of original rows or columns. Here the sampling procedure is to choose columns with a probability distribution over the Euclidean norm of the rows of the top k right singular vectors. The probability distribution is expressed as :

$$p_i = \frac{\beta |V_{k_{(i)}}|^2}{\sum_{j=1}^{n} |V_{k_{(j)}}|^2}$$

The sampling procedure introduces the notion of doing importance sampling over the best possible leverage scores.

Two sampling procedures are given, EXACTLY(ℓ), which picks exactly ℓ columns with probability p_i in ℓ $i.i.d$ trials, and EXPECTED(ℓ), where at most ℓ columns are chosen in expectation of a probability $\min\{1, \ell * p_i\}$. The rescaling factor of columns chosen is $1/\sqrt{(\ell * p_i)}$ and $1/\min\{1, \sqrt{(\ell * p_i)}\}$. The complete EXACTLY($\ell$) algorithm is given in algorithm 3.1.3.

For an input matrix A, a sketch matrix B is computed that has the sampled columns $\ell = O(k^2 \log(1/\delta)/\varepsilon^2)$, and BB^+A is the projection of A on the subspace spanned by the sampled columns.

Algorithm 3.1.3 SUBSPACE SAMPLING

Input: $\ell, \epsilon \in (0, 1], k, A \in \mathbb{R}^{d \times n}$
$B_0 \leftarrow$ all zeros matrix $\in \mathbb{R}^{d \times ell}$
Compute SVD(A) $A = USV^T$
Get top$k$ vectors V_k
for $i \in [n]$ **do**
    Compute $total = \sum_{j=1}^{n} |V_{k_{(j)}}|^2$
for $i \in [n]$ **do**
    Compute probability $p_i = \beta |V_{k_{(i)}}|^2 / total$, for each column a^i
    for $j \in [\ell]$ **do**
        Insert a^i into B by sampling with replacement
        Each column is rescaled as $B^j = a^i / \sqrt{(\ell * p_i)}$
return B

The algorithm gives relative error bounds on the Frobenius norm as follows:

$$\|A - BB^{+}A\|_F \leq (1+\varepsilon)\|A - A_k\|_F$$

To the best of our knowledge, this is the first algorithm to give a relative error approximation for computing a low-rank approximation. The running time is dominated by the SVD step and the algorithm needs two passes over data, one to compute SVD and the other to compute the sketch matrix. The running time for the algorithm is $O(SVD(A_k) + nnz(A) + C)$. Since SVD is the building block for SUBSPACE SAMPLING, a streaming version of this algorithm is difficult to accomplish, but can be achieved by using Frequent Directions and is an open theoretical question. The space required by the algorithm is $O(nnz(A) + nnz(B))$.

3.1.4 Determinisitc Leverage Score Sampling

Recently, D. Papailiopoulos *et al.* [46] proposed an "energy"-based deterministic leverage score sampling procedure. They consider computing the top k right singular vectors of matrix A and then computing the leverage scores. They sort the leverage scores and sample columns of A corresponding to the largest leverage scores.

By "energy"-based, they do not specify the number of samples, but they continue sampling until the sum of leverage scores exceeds a threshold θ. We note here, that this only works for datasets whose leverage scores follow a power-law distribution. Also one of the drawbacks of their algorithm is that they might end up sampling all columns of matrix A.

Algorithm 3.1.4 Deterministic Leverage Score Sampling

Input: $\ell, \epsilon \in (0,1], k, A \in \mathbb{R}^{d \times n}$
$B_0 \leftarrow$ all zeros matrix $\in \mathbb{R}^{d \times ell}$
Compute SVD(A) $A = USV^T$
Get top$k$ vectors V_k
for $i \in [n]$ **do**
    Compute $lev\_scores_i^{(k)} = \|[V_k]_i, :\|^2$
    $lev\_scores_i^{(k)}$ be sorted
$\theta = k - \epsilon$
for $j \in [\ell]$ **do**
    Insert a^i into B for $lev\_scores_i^{(k)}$
return B

D. Papailiopoulos *et al.* provide a theoretical guarantee that by setting $\theta = k - \varepsilon$, we can achieve the relative error bound $\|A - CC^{+}A\|_{F,2}^{2} \leq (1-\varepsilon)^{-1}.\|A - A_k\|_{F,2}^{2}$. Note that $\varepsilon \in (0, 1/2)$ gives a $(1+2\varepsilon)$ guarantee. The main steps of the algorithm are given in 3.1.4. Notice the difference in step 8 where energy based parameter is invoked for column selection. To bound the number of samples, they consider a special case of leverage scores following a power-law decay with exponent $\alpha_k = 1 + \mu$, and show that the number of samples (ℓ) is bounded by:

$$\ell = \max\{\left(\tfrac{2k}{\varepsilon}\right)^{\frac{1}{1+\mu}} - 1, \left(\tfrac{2k}{\mu\varepsilon}\right)^{\frac{1}{\mu}} - 1, k\}$$

Then they can achieve a $(1-\varepsilon)^{-1}$ relative error bound. The running time and space constraints are the same as the above SUBSPACE SAMPLING algorithm. It is clear that one can speed up their algorithm via randomized SVD algorithms to get a fast approximation to V_k. One way of doing this that is mentioned in their paper is to use the Frequent Directions algorithm, in place of the exact computation of SVD, which will the reduce running time from $O(nd^2)$ to $O(ndk/\varepsilon)$.

3.1.5 Reservoir Sampling

In reservoir sampling, we want to select a random sample of ℓ items with or without replacement from a collection of n items, where each item arrives one after the other, and a reservoir is maintained of the ℓ samples seen so far. The generic algorithmic steps for reservoir sampling are given in algorithm 3.1.5. First, we fill up the reservoir B with the first ℓ rows of the data matrix, and simulatenously compute a prioritry/rank for each row as shown in steps 4 and 5 of algorithm 3.1.5.

The next set of steps describes whether we should select a row after ℓ and include it in the reservoir or not.The row with least prioirty is recorded as the threshold τ from the rows in the reservoir B seen so far, as we want to keep rows in the reservoir that are greater than the threshold τ computed so far. For the rows of the matrix from $\ell + 1$ until the last row, we compute priority for each row and compare it with τ.

If the priority of a row is greater than τ, then the row corresponding to τ is replaced with the current row between $\ell + 1$ and n. Note the SELECT algorithm used in LinearTime SVD makes it a reservoir sampling algorithm.

Algorithm 3.1.5 Generic Reservoir Sampling

Input: $\ell, A \in \mathbb{R}^{n \times d}$
$B_0 \leftarrow$ all zeros matrix $\in \mathbb{R}^{\ell \times d}$
for $i \in [\ell]$ **do**
    Compute priority p_i for row a_i
    Insert row a_i in reservoir B as (p_i, a_i)
$\tau =$minimum priority item in reservoir B
for $j \in [\ell + 1..n]$ **do**
    Compute priority p_j for row a_j
    if $(p_j > \tau)$ **then**
        Replace item corresponding to τ with (p_j, a_j)
return B

Importance sampling algorithms such as SUBSPACE SAMPLING, LINEAR TIME SVD, and DETERMINISTIC LEVERAGE SCORE SAMPLING are weighted sampling with replacement. One of the concerns with this set of algorithms is the higher likelihood of duplicating heavy items in the sketch matrix. The sketch matrix generated by importance sampling algorithms captures relatively less information of the light items. In the following sections, we look at weigh-sensitive algorithms without replacement that have not been used for computing a low-rank approximation and were meant for estimating arbitary subset sums.

3.1.6 Priority Sampling (New)

PRIORITY SAMPLING [21] is a weight-sensitive sampling without replacement technique. It computes a sample of size $\ell + 1$ for estimating the weight of arbitary subsets for a given collection of weighted item. A row of a matrix with its Euclidean norm is a weighted item and PRIORITY SAMPLING is used to compute a sketch matrix of $\ell + 1$ rows, which best approximate the original matrix. Duffield *et al.* [21] conjectured that total variance is minimal among the $\ell + 1$ unbiased estimators, which was proven separately by [50].

PRIORITY SAMPLING generates a priority $q_i = w_i/u_i$ for each item a_i where $u_i = uniform - random(0, 1)$. It puts the first $\ell + 1$ items of the stream in b and sets the threshold τ to be the smallest priority. Future items of the stream will be placed in B, only if their priority is larger than τ. After each insertion, τ will get recomputed. This way the sample B will consist of the ℓ highest priority items. Each sampled item a_i in B gets a weight estimate $\hat{w}_i = \max\{w_i, \tau\}$, whereas nonsampled items get weight estimate $\hat{w}_i = 0$.

Algorithm 3.1.6 Priority Sampling

Input: $\ell, A \in \mathbb{R}^{n \times d}$
$B_0 \leftarrow$ all zeros matrix $\in \mathbb{R}^{\ell+1 \times d}$
for $i \in [\ell]$ **do**
  Generate an independent uniform $u_i = uniform - random(0, 1)$
  Compute priority for row a_i as $q_i = w_i/u_i$
  Insert row a_i in reservoir B as (q_i, a_i)
$\tau =$ $(\ell + 1)$th priority item in reservoir B
for $j \in [\ell + 1..n]$ **do**
  Compute priority q_j for row a_j
  if $(q_j > \tau)$ **then**
    Replace item corresponding to τ with (q_j, a_j)
for $i \in [\ell]$ **do**
  Rescale $\hat{w}_i = \max(w_i, \tau)$
return B

We refer the reader to the original paper [21], which has an elegant proof for the estimate being unbiased in section 2. PRIORITY SAMPLING fits in nicely in the streaming model and is straightforward to implement. It is also possible to capture which rows are part of the sketch matrix using constant additional space. Since PRIORITY SAMPLING uses a priority queue for updates, the running time for PRIORITY SAMPLING is $O(nnz(A) + I * \log(\ell))$ where I is the expected number of insertions. The space needed for PRIORITY SAMPLING is $O(\ell d)$.

3.1.7 Varopt Sampling (New)

VARIANCE OPTIMAL SAMPLING was introduced by Cohen *et al.* [12]. In this algorithm, the scheme is constructed to satisfy three goals, which come from statisitics.

- Ipps (Inclusion probability proportional to size) :

  Each item is selected with a probability $p_i = \min(1, w_i/\tau_k)$ where i corresponds to an item, w_i corresponds to the weight of item i, and τ_k is the threshold value for k items to be in the sample, and is defined as:

  $$\sum_{i \in [n]} \min(1, w_i/\tau_k) = k$$

  If $\tau_k = 0$, then $k \geq n$ and all items are sampled. Each item i gets weight as $\max(w_i, \tau_k)$

- At most, k items are present in the sample estimate
- No positive covariances

VARIANCE OPTIMAL SAMPLING is similar to PRIORITY SAMPLING, but the computation of threshold τ to take a decision of which items to keep in the reservoir is different, which we describe below. We have a reservoir of ℓ items with weights as $w_{i=[1...k]}$ for each item. Let these weights be in increasing sorted order. Compute the largest number t such that $w_{(t)} \leq \tau$. These can be expressed as:

$$\begin{aligned} \mathrm{w}_{(t)} \leq \tau \Leftrightarrow k + 1 - t + \left(\sum_{x \leq t} w_{(x)}/w_{(t)} \right) \geq k \\ \leq \tau \Leftrightarrow \left(\sum_{x \leq t} w_{(x)}/w_{(t)} \right) \geq t - 1 \end{aligned}$$

After computing t we can compute τ as,

$$\tau = \left(\sum_{x \leq t} w_{(x)} \right) / \left(t - 1 \right) \tag{3.1}$$

We need to drop an item from the reservoir, for which we draw a uniform random number $r \in (0, 1)$, and search for the smallest $d \leq t$ such that,

$$\sum_{x \leq d} \left(1 - w_{(x)}/\tau \right) \geq r \Leftrightarrow \left(d\tau - \sum_{x \leq d} w_{(x)} \right) \geq r\tau \tag{3.2}$$

Since we need item weights to be in sorted order to compute threshold τ, we maintain a priority queue of L items. Items with weight $w > \tau$ are maintained in the priority queue L. We maintain another list T, where all items have adjusted weight τ. Together $L \cup T$ make the ℓ items for the sketch matrix. When a new item i arrives in the stream, we computzae a new threshold τ_{new} based on the threshold computation above. We also maintain another list X where we capture items whose weight is less than the new threshold. If the weight of the new item i is less than the existing threshold τ, then we move the item i to the list X, or add the item to L.

Now that we have the new threshold τ_{new}, we need to remove items from L that are less than τ_{new}. We use the priority queue to accomplish this, as it gives us the smallest item, in L and move all items whose weight is less than τ_{new} from L to X. The number of items in $L + T + X$ is still $\ell + 1$ and not ℓ as we need. To drop one item, we use the dropping item

procedure described in equation 3.2. If the condition in equation 3.2 is not satisified, we randomly delete an element from T. After the deletion procedure is completed, we move all items X from T and continue until all rows are processed in the stream.

It is worth noting here that all the items in list L maintain their original weight and this scheme gives us the actual rows from L, which gives us interpretability of the rows similar to SUBSPACE SAMPLING and DETERMINISTIC LEVERAGE SCORE SAMPLING.

For the complete algorithm, we refer the reader to section 4.3 in [12]. The running time for a randomly permuted stream with n datapoints for VARIANCE OPTIMAL SAMPLING is $O(nnz(A) + \ell(\log(\ell)(\log n)))$ in expectation. The overall space is the same as PRIORITY SAMPLING.

3.1.8 Weighted Reservoir Sampling (New)

WEIGHTED RESERVOIR SAMPLING W/O REPLACEMENT [22] does sampling without replacement proportional to rows weights. As algorithm 3.1.7 processes the stream, it assigns each item a key $k_i = u_i^{1/w_i}$ where $u_i = uniform - random(0, 1)$ and then simply keeps the top ℓ items ordered by their keys. Initially, WEIGHTED RESERVOIR SAMPLING W/O REPLACEMENT puts first ℓ items into the sample set B and sets the current threshold τ to the smallest key in B. Then for each item a_j $\forall j = s + 1, \cdots, n$ it generates k_j and if k_j is larger than τ, the item gets replaced with the item of the smallest key in B and τ gets recomputed. Each sampled item a_i in B gets a weight estimate $\hat{w}_i = W/s$ where W is the total weight of the stream.

The running time and space needed are similar to the PRIORITY SAMPLING without replacement method. Note that by using this technique and having ℓ reservoir sampler each sampling one item, we can have a weighted reservoir sampler with replacement.

Algorithm 3.1.7 Weighted Reservoir Sampling

1: **Input:** $\ell, A \in \mathbb{R}^{n \times d}$
2: $B_0 \leftarrow$ all zeros matrix $\in \mathbb{R}^{\ell+1 \times d}$
3: **for** $i \in [\ell]$ **do**
4: Generate an independent uniform $u_i = uniform - random(0, 1)$
5: Compute priority for row a_i as $q_i = u_i^{1/w_i}$
6: **return** B

3.2 Random Projections

All random projection techniques are based on the seminal work of William B. Johnson and Joram Lindenstrauss [29], which governs low-distortion embeddings of points from high-dimensional space into low-dimensional Euclidean space. The Johnson-Lindenstrauss lemma states that a set of n points can be embedded in a space of (lower) dimension $O(\log n/\varepsilon^2)$ and preserve all pairwise distances within a factor of $(1+\varepsilon)$ with high probability. The formal definition of the lemma is as follows.

3.2.1 JL Lemma

Lemma 3.2.1. *Johnson-Lindenstrauss (JL) lemma [13]:*

Having a set of n points $a_1, \ldots, a_n$ in $\mathbb{R}^d$, there exists a linear mapping $f : \mathbb{R}^d \to \mathbb{R}^k$ with $k > \frac{9 \log n}{\varepsilon^2 - \varepsilon^3} \log 1/\delta$ such that it preserves all pairwise distances up to distortion ε with high probability at least $1 - \delta$:

$$\forall i, j \; (1-\varepsilon)\|a_i - a_j\|^2 \leq \|f(a_i) - f(a_j)\|^2 \leq (1+\varepsilon)\|a_i - a_j\|^2$$

Any function f that satisfies this inequality is called a Johnson-Lindenstrauss transform, $JLT(\varepsilon, \delta, n)$.

This lemma is essential to many applications such as approximating a nearest neighbors search, manifold learning, dimension reduction, and compressed sensing. After the Johnson-Lindenstrauss lemma, many variants and low-distortion embedding techniques emerged, all of which share the same core idea: They select a matrix $S \in \mathbb{R}^{\ell \times n}$ from a probability distribution such that the distance between two arbitrary points a_i and a_j is preserved with a high probability. Therefore the product Ax for any vector $x \in \mathbb{R}^d$ gives the linear mapping $f(x)$. They get the union bound on failure probability of all $x \in \mathbb{R}^d$, which translates to all $\binom{n}{2}$ pairwise distance in a set of n points, and at the end choose ℓ such that the failure probability is a small constant.

Algorithm 3.2.8 Generic Random Projections

1: **Input:** $\ell, A \in \mathbb{R}^{n \times d}$
2: Generate subspace embedding matrix $S \in \mathbb{R}^{\ell \times n}$
3: Compute $B = S * A$
4: **return** B

Here, the matrix S is a subspace embedding matrix. If $V \subseteq \mathbb{R}^n$ (linear) has dimension d, then $S \in \mathbb{R}^{\ell \times n}$ is an ϵ subspace embedding for V, such that for every $x \in V$, $\|Sx\| = (1 \pm \epsilon)\|x\|$.

The intuition is to have a subspace embedding that preserves the norm of any vector in the row space or column space of the original matrix up to $1 \pm \varepsilon$. The SVD of the matrix $B \in \mathbb{R}^{\ell \times d}$, which is the product A and S, approximates the original data matrix A as does its singular vectors.

Different random projection methods use different subspace embedding matrices. Dasgupta *et al.* [13] gave a simple proof, using a Gaussian random matrix for performing embedding in lower-dimensional space, and Achlioptas [3] showed a random matrix having entries $e \in (-1, 0, 1)$ also does the same. Note that the latter construction preserves sparsity. The above-mentioned constructions need $O(\ell d)$ time per datapoint to get lower-dimensional embedding $\mathbb{R}^k$. We look at the running time again when we are looking at FAST JLT.

The generic algorithm steps are given in algorithm 3.2.8.

Using a dense subspace embedding matrix and multiplying it with the original data matrix A is time consuming and can be worse than computing the SVD of the original data matrix. In the following sections we look at subspace embedding matrices, which have a specifc structure for performing matrix multiplication quickly.

3.2.2 Random Projections - Sarlos 06

In 2006, Sarlos [49] gave the first relative error bound algorithm in this area. He used a rademacher matrix (sign matrix) S, in which each entry of the matrix is a $i.i.d$ zero mean +1,-1 value. Matrix $S \in \mathbb{R}^{\ell \times n}$ is a Johnson-Lindenstrauss transform that projects datapoints from high-dimensional space $\mathbb{R}^n$ to low-dimensional space $\mathbb{R}^\ell$ where $\ell = O(k/\varepsilon + k \log k)$. The sign matrix can be seen as a subspace embedding.

Sarlos showed projecting A on to rowspace of AS^T achieves a relative error bound $\|A - \pi_{AS^T,k}(A)\|_F \leq (1 + \varepsilon)\|A - A_k\|_F$.

The running time for Sarlos's algorithm is $O(\ell d)$ per datapoint and needs $O(\ell d)$ space. We note here that in our results, RANDOM PROJECTION refers to Sarlos's algorithm using a sign matrix. Notice that doing matrix multiplication with a sign matrix is very fast, as

we can perform addition or subtraction operations when each new datapoint comes in the stream.

3.2.3 Fast JLT

Sarlos also mentions using a Fast Johnson-Lindenstrauss transform (FAST JLT), introduced by Ailon and Chazelle [5], to compute a low-rank approximation. Matrix multiplication can be sped up by using FAST JLT, which uses the idea of the Hadamard transform to construct the sketch matrix quickly.

The distribution given by Ailon and Chazelle is a matrix that is a product of three matrices given as: $S = \frac{1}{\sqrt{(\ell n)}} PHD$

Here $P \in \mathbb{R}^{\ell \times n}$ is a random sparse matrix whose $i.i.d$ entries are 0 with probability $1-q$ or $N(0, q^{-1})$ with probability q where $q = \min\{1, \Theta(\frac{\log^2 n}{d})\}$. $H \in \mathbb{R}^{n \times n}$ is a Hadamard matrix where $H_{ij} = n^{-1/2}(-1)^{\langle i-1, j-1 \rangle}$ and $\langle i, j \rangle$ is the dot product of the $m-$ bit vectors i, j in binary form. Here n needs to be a power of 2. If it is not a power of 2, pad zeros to the Hadamard matrix. The following is an example of Hadamard matrix:

$$H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$D \in \mathbb{R}^{n \times n}$ is a diagonal matrix with random signs on its diagonal. If the data matrix is very sparse, then multiplying it by a sign matrix will distort the heavy mass vector (row). Instead, if we multiply a given row in such a way that we spread the mass of the row, while also preserving norms, we should get a better approximation. The Fourier transform can be used to do this, because of Heisenberg's uncertainity principle, which states that, given a vector x and its spectrum, both cannot be sparse.

If the input is uniformly distributed, then the Fourier transform will be unneeded. Hence we also need a random sign diagonal matrix to get a good spread of a vector. In addition, multiplication by a Hadamard transform can be done in $O(n \log(n))$ time. Notice that a Hadamard transform construction is deterministic, and the matrix S can be constructed independent of the original matrix.

One minor issue with the Hadamard matrix is its requirement of the order of the matrix to be a power of 2. This can be done in streaming enviroment, by just storing the matrix S in memory and applying the entries corresponding to each datapoint as it comes.

Here the Hadamard transform would take $O(n \log(n))$ time and $O(\ell^3)$ [38] to apply the sparse projection matrix P to the Hadamard transform. Hence applying the transform to any vector x takes $O(n \log(n) + \ell^3)$ time in expectation per vector. The running time bound is not optimal in comparison with dense random projection, for, e.g., the Gaussian random matrix as the distribution matrix. If $k \in O(\log(n))$, then $O(\ell n)$ becomes $O(n \log(n))$ for the Gaussian matrix, whereas for FAST JLT it will be $O(n \log(n) + \log^3(n))$. There has been work to improve the ℓ^3 factor, but we do not include it in this manscript as the constructions proved hard to implement efficiently.

3.2.4 Toeplitz Matrix

Hinrichs and Johnson [28] give a variant of JOHNSON-LINDENSTRAUSS lemma using partial Circulant/Toeplitz matrices. They give a proof where embedding dimension k is bounded as $O(\varepsilon^{-2} \log^3 n)$ instead of the bound of JOHNSON-LINDENSTRAUSS, $k = O(\log n/\varepsilon^2)$. The construction is straightforward and runs in $O(d \log(d))$ time.

We used a TOEPLITZ matrix to construct a subspace embedding. A TOEPLITZ matrix is a diagonal-constant matrix. The entries along all the diagonals have constant entries. An example of a TOEPLITZ matrix is as follows:

$$T = \begin{bmatrix} a & b & c & d \\ e & a & b & c \\ f & e & a & b \\ g & f & e & a \end{bmatrix}$$

The construction follows from [43] and is given below: Construct a random Bernoulli vector $v = (v_0, v_1, ...v_{2d-1})$. Construct a TOEPLITZ matrix $(T_v)_{ij} = v_{((i+j) \mod d)}$ and have a diagonal matrix D with random signs following the FAST JLT discussion. The subspace embedding is given as $T_v D/\sqrt{(k)}$. Even though TOEPLITZ does not have optimal bounds, in practice it gives accuracy as good as RANDOM PROJECTION and CW TRANSFORM - 09 subspace embedding and runs as fast, while having a ℓ^3 factor improvement over FAST JLT in its running time.

3.2.5 CW TRANSFORM - 09

In 2009, Clarkson and Woodruff [10] proposed several algorithms to address this problem in the row-wise update model and turnstile model.

Assuming $A \in \mathbb{R}^{n\times d}$ is given a row at a time, the authors proposed a one-pass algorithm that uses a sign matrix $S \in \mathbb{R}^{\ell\times n}$ where $\ell = O(k/\varepsilon \log 1/\delta)$ (an improvement over the work of Sarlos). SA transforms datapoints to a lower dimension and projecting A onto SA gives the $(1+\varepsilon)$ relative error bound. The interesting point about their algorithm is that it does not need an additional pass for computing the projection $\pi_{SA}(A)$ so everything is getting computed in exactly one pass over the data because the projection is dependent on values of two matrices SA and SAA^T only, and these two matrices can get computed in one pass over the data. The best rank$-k$ approximation of the projection matrix satisfies the $(1+\varepsilon)$ relative error bound. This algorithm takes $O(n^2 \log(n)) + npoly(k/\varepsilon)$ time, which may be faster than Sarlos's algorithm for dense matrices and large k.

3.2.6 Sparse Random Projections

In 2013, Clarkson and Woodruff [11] gave CW TRANSFORM - 13 where they proposed a new algorithm that consturcts matrix $\hat{A} = LDW^T$ and improves the running time to $O(nnz(A)) + npoly(k/\varepsilon(\log n))$. They construct a matrix R as $R = \Pi\bar{R}$ where $\in \mathbb{R}^{t'\times t}$ is a sampled randomized Hadamard matrix with $t = O(r^2 \log^6(r/\varepsilon) + r/\varepsilon)$ and $t' = O(r/\varepsilon \log(r/\varepsilon))$ with $r = rank(A)$ and $\bar{R} \in \mathbb{R}^{t\times n}$ is a sparse embedding matrix. They compute AR^T and an orthogonal basis U for column space of AR^T. After that, they compute SU and SA, for S the product of a $v' \times v$ SRHT matrix with a $v \times n$ sparse embedding where $v = \theta(\varepsilon^{-4}k^2 \log^6(k/\varepsilon))$ and $v' = \theta(\varepsilon^{-3}k \log^2(k/\varepsilon))$. They compute the SVD of $SU = U\Sigma V^T$, and SVD of $\hat{U}DW^T = V\Sigma^-[U^T SA]_k$. They return $L = U\hat{U}, D, W$.

The computation of the above procedure is complicated with the construction bounds that are presented. Instead we use the sparse embedding matrix construction given by Clarkson and Woodruff and use our standard projection bounds to compute the low-rank approximation.

The idea of the sparse embedding matrx is not new entirely. It is a simple count sketch construction where in each column of the matrix we randomly put ±1 in one of the rows. Hence every column has one nonzero entry. The bounds given were improved by [40, 44] to $\ell = (d^2/\varepsilon^2)$.

CW TRANSFORM - 13 has $s = 1$ for $\ell = O(d^2/\epsilon^2)$. An example of the embedding matrix is as follows :

$$S = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 \end{bmatrix}$$

Nelson and Nguyen [44] gave another construction OSNAP, which is similar to the above construction, but instead of having one nonzero entry, the column is divided into blocks of size ℓ/s, where each block has one nonzero entry as ± 1 randomly. s is the number of sparse entries we need in a column. If $s = 1$, this is the same as the CW TRANSFORM - 13. An example of OSNAP embedding is as follows:

$$S = \begin{bmatrix} 0 & -1 & 0 & -1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 1 & -1 \end{bmatrix}$$

Also another sparse construction we used is CZ TRANSFORM - 14 [52]. Here the sparse embedding matrix of CW TRANSFORM - 13 is multiplied by a diagonal matrix D where each entry on the diagonal is a reciprocal of an exponential random variable $E_1....E_n$.

An example of CZ TRANSFORM - 14 embedding is as follows :

$$S = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1/E_1 & 0 & 0 & 0 \\ 0 & 1/E_2 & 0 & 0 \\ 0 & 0 & 1/E_3 & 0 \\ 0 & 0 & 0 & 1/E_4 \end{bmatrix}$$

3.3 Frequent Direction: Variants

Here, we look at the FREQUENTDIRECTIONS algorithm (given by Liberty [37]) and its variants [26]. FREQUENTDIRECTIONS is inspired by the Frequent Items algorithm also known as Misra-Gries [42]. This algorithm is deterministic and computes the sketch in only one pass.

3.3.1 FREQUENTDIRECTIONS

For counting the top ℓ frequent items in a given stream, Frequent Items maintains ℓ counters for item frequency and ℓ labels for items. For each new item, if the item is already seen, then its corresponding counter is incremented. If the item is completely new, then we look to see if there is any unused counter (i.e., count = 0). If such a counter is present, then we associate the given counter with this item by using one of the ℓ labels and increment its

count by 1; otherwise we decrement each of the current counters by 1 and delete an item if the counter is 0.

FREQUENTDIRECTIONS extends this concept to matrices. Given a $n \times d$ matrix A, it maintains $\ell \times d$ sketch matrix B where $\ell \ll d \ll n$. Each row of A fills up B until ℓ, after which we compute SVD of B. The SVD step gives us orthogonal vectors. Here the vectors represent the directions and we remove one direction, i.e., make a vector zero by subtracting the last singular value from all other singular values. This subtraction makes space for a new row to come in, and we repeat these steps until all rows are processed. The singular vectors of sketch matrix B are analogous to the labels used in Frequent Items, and the singular value corresponds to the counters. The generic algorithm is given in algorithm 3.3.9. The ReduceRank step is where the algorithms will be different, but the rest of the steps remain the same.

The algorithm given by Liberty in [37] subtracts the singular at $\ell/2$, which makes space for $\ell/2$ new data points. This is faster, but gives slighlty weaker error guarantees. The algorithm with a stronger error guarantees subtracts the smallest singular value from all the singular values for sketch matrix B. The ReduceRank modification is given in algorithm 3.3.10.

We restate the facts given by [26] and [37] below:

- Fact 1: For any unit vector x we have $\|Ax\|^2 - \|Bx\|^2 \geq 0$.
- Fact 2: For any unit vector x we have $\|Ax\|^2 - \|Bx\|^2 \leq \Delta$.
- Fact 3: $\|A\|_F^2 - \|B\|_F^2 \geq \alpha\Delta\ell$.

These facts guarantees error bounds for FREQUENTDIRECTIONS and its variants given in [25]. The error bounds for FREQUENTDIRECTIONS are as follows:

$$\text{For } \alpha \in (0,1), \text{ and } \alpha = 1 \text{ for FREQUENTDIRECTIONS,}$$
$$\|A^TA - B^TB\|_2 \leq \|A - A_k\|_F^2 / (\alpha\ell - k)$$
$$\|A - \pi_{B_k}(A)\|_F^2 \leq \|A - A_k\|_F^2 \alpha\ell / (\alpha\ell - k)$$

Algorithm 3.3.9 (Generic) Frequent Direction Algorithm

Input: $\ell, \alpha \in (0,1], A \in \mathbb{R}^{n \times d}$
$B_0 \leftarrow$ all zeros matrix $\in \mathbb{R}^{\ell \times d}$
for $i \in [n]$ **do**
  Insert a_i into a zero valued rows of B_{i-1}; result is B_i
  if (B_i has no zero valued rows) **then**
    $[U, S, V] \leftarrow \mathsf{svd}(B_i)$
    $C_i = SV^T$ # Only needed for proof notation
    $S' \leftarrow ReduceRank(S)$
    $B_i \leftarrow S'V^T$
return $B = B_n$

Algorithm 3.3.10 FREQUENTDIRECTIONS Algorithm

$\delta_i \leftarrow \sigma_\ell^2$
return $\mathsf{diag}(\sqrt{\sigma_1^2 - \delta_i}, \ldots, \ldots, \sqrt{\sigma_\ell^2 - \delta_i})$

3.3.2 ISVD

The ReduceRank procedure ISVD throws away the last singular value to make space for a new row. For ISVD, algorithm 3.3.9 keeps $\sigma'_j = \sigma_j$ for $j < \ell$ and sets $\sigma'_\ell = 0$.

3.3.3 PARAMETERIZED FREQUENT DIRECTIONS

In FREQUENTDIRECTIONS we subtract the smallest singular value from all other singular values. The largest singular values are the ones that correspond best to the signal in the data matrix. Subtracting the smallest singular value from the largest singular value shrinks the signal magnitude in every iteration, which leads to signal loss. Instead in [25], we showed a variant α − FREQUENTDIRECTIONS, where α decides which row index should start the singular value subtraction and $\alpha \in (0,1)$. To understand α, for ISVD the value of $\alpha = 0$, as we always remove the last singular value, whereas for FREQUENTDIRECTIONS it is 1, as we subtract from all values. Its running time is same as FREQUENTDIRECTIONS and is deterministic. The ReduceRank step is given in algortihm 3.3.11.

Algorithm 3.3.11 α−Frequent Direction Algorithm

$\delta_i \leftarrow \sigma_\ell^2$
return $\mathsf{diag}(\sigma_1, \ldots, \sigma_{\ell(1-\alpha)}, \sqrt{\sigma_{\ell(1-\alpha)+1}^2 - \delta_i}, \ldots, \sqrt{\sigma_\ell^2 - \delta_i})$

Given an input matrix $A \in \mathbb{R}^{n \times d}$, PARAMETERIZED FREQUENT DIRECTIONS with parameter ℓ returns a sketch $B \in \mathbb{R}^{\ell \times d}$ that satisfies

$$0 \leq \|Ax\|^2 - \|Bx\|^2 \leq \|A - A_k\|_F^2/(\alpha\ell - k)$$

and projection of A onto B_k, the top k rows of B satisfy

$$\|A - \pi_{B_k}(A)\|_F^2 \leq \frac{\alpha\ell}{\alpha\ell - k}\|A - A_k\|_F^2.$$

This variant satisfies the error bounds given for FREQUENTDIRECTIONS by following the three facts mentioned above and performs better in terms of error in practice due to the α tweak.

3.3.4 Space Saving Directions and Compensative FD

There is a variant proposed by [41] similar to Misra-Gries called Space Saving. In Space Saving, when a new item comes in that has not been seen in the stream so far and all labels are occupied, then we replace the label with the least count with the new item and increment its counter. We apply the same concept to the matrix setting and call it SPACESAVING DIRECTIONS.

After the SVD step, instead of subtracting the last singular value, we add the last singular to the second to the last singular value, and make one of the values 0, which is akin to the increment step in Space Saving for Frequent Items. Also, the second to the last vector is changed to the last vector for the analogous label step. This variant also follows the three facts and hence has the error guarantees asymptotically as FREQUENTDIRECTIONS. This variant does not do as well as PARAMETERIZED FREQUENT DIRECTIONS but matches or outperforms FREQUENTDIRECTIONS.

Another variant in Space Saving direction, is instead of just replacing the second to the last vector, add the second to the last vector and the last vector. Since this is an orthogonal subspace, vector addition will lead to a resulting vector in the same subspace which does marginally better than SPACESAVING DIRECTIONS; hence, we do not report it.

Another variant is COMPENSATIVE FREQUENT DIRECTIONS in which we sum the singular values we subtracted in each iteration. These values give the total mass that we have subtracted from the stream. When we get the final sketch matrix B after all rows are processed, we distribute the weight we have subtracted back to the matrix. Asymptotically

the above mentioned algorithm also matches FREQUENTDIRECTIONS bounds, and does only as well as FREQUENTDIRECTIONS in practice.

3.3.5 FREQUENTDIRECTIONS: Running Time

FREQUENTDIRECTIONS and its variants, PARAMETERIZED FREQUENT DIRECTIONS, have the highest running time, which is linear as the sketch size increases. Even though FREQUENTDIRECTIONS gives the best accuracy, running time is still a major bottleneck. The high running time is due to the high number of SVD computations that we do. To improve the running time, we propose subtle modifications to the FREQUENTDIRECTIONS and PARAMETERIZED FREQUENT DIRECTIONS algorithm, which can be used for any variant including ISVD.

3.3.6 Tweak FREQUENTDIRECTIONS (New)

In the original FREQUENTDIRECTIONS algorithm given by Liberty [37], we zero out half of the sketch matrix B, by subtracting the singular value at $\ell/2$, which is fixed and deterministic, but Liberty remarks in section 2.1 of [36], for some $c \in [1/10, 9/10]$, if we subtract the singular value at $c\ell$, then the bounds still hold, but are slightly weaker.

The singular values of Lena given in Figure 1.1 had a big drop-off after the first 100 singular values, and we look at rank-100 approximation instead of the original matrix. Similarly, whenever we compute the SVD of the sketch matrix B, we look for the index that has the highest drop-off. Starting from $c = \ell...1$, we compute the minimum value of the ratio of the singular value at c over the singular value at $c-1$ as given in Step 1 in algorithm 3.3.12. We ensure that $c \in [1/10, 9/10]$, and if c is not in the range, then we revert to Liberty's step, of zeroing half of the sketch.

3.3.7 Tweak PARAMETERIZED FREQUENT DIRECTIONS (New)

The algorithm for PARAMETERIZED FREQUENT DIRECTIONS given in algorithm 3.3.11 is slow, as it is able to make space at least for one new row and not much more. One way to speed it up is to apply Liberty's trick of halving the sketch.

Instead of subtracting the last singular value from $(1-\alpha)\ell \dots \ell$, we subtract the singular value at $(\ell - t)$ where $t = \ell\alpha/2$, from all the singular values between $[\ell - 2t \dots \ell - t]$.

Algorithm 3.3.12 Tweak FD Algorithm

$k = \min(\sigma_c/\sigma_{c-1})$
$0.1 \leq k \leq 0.9$
$\delta_i \leftarrow \sigma_k$
return diag$(\sqrt{\sigma_1^2 - \delta_i}, \ldots, \ldots, \sqrt{\sigma_\ell^2 - \delta_i})$

The tweaked subtraction gives a very good speed-up as we will see in the results later and also has strong error guarantees. Facts 3.3.1 and 3.3.1, for Tweak PARAMETERIZED FREQUENT DIRECTIONS, hold by the same argument as in Lemma 2.1 in [25]. Briefly, each singular value is decreased, and by at most δ_i at each step. We next prove Fact 3.3.1.

Lemma 3.3.2. *For any $\alpha \in (0, 1]$, $\|A\|_F^2 - \|B\|_F^2 \geq \frac{\alpha}{2}\Delta\ell$, proving Fact 3.*

Let $t = \ell\alpha/2$

Expand, $\|C_i\|_F^2 = \sum_{j=1}^{\ell} \sigma_j^2$ to get,

$\|C_i\|_F^2 = \sum_{j=1}^{(\ell-2t)} \sigma_j^2 + \sum_{j=(\ell-2t)}^{\ell-t} \sigma_j^2 + \sum_{j=(\ell-t)}^{\ell} \sigma_j^2$

$\|C_i\|_F^2 = \sum_{j=1}^{(\ell-2t)} \sigma'^2_j + \sum_{j=(\ell-2t)}^{\ell-t} (\sigma'^2_j + \delta_i) + \sum_{j=(\ell-t)}^{\ell} \sigma_j^2$

$\|C_i\|_F^2 \geq \|B_i\|_F^2 + t\delta_i.\|a_i\|^2 = \|C_i\|_F^2 - \|B_{i-1}\|_F^2 \geq (\|B_i\|_F^2 + \frac{\alpha}{2}\ell\delta_i) - \|B_{i-1}\|_F^2$

and summing over i we get, $\|A\|_F^2 = \sum_{i=1}^{n} \|a_i\|^2 \geq \sum_{i=1}^{n} \|B_i\|_F^2 - \|B_{i-1}\|_F^2 + \frac{\alpha}{2}\ell\delta_i = \|B\|_F^2 + \frac{\alpha}{2}\ell\Delta$. Subtracting $\|B\|_F^2$ from both sides completes the proof. □

Algorithm 3.3.13 Tweak PARAMETERIZED FREQUENT DIRECTIONS Algorithm

$t = \ell\alpha/2$
$\delta_i \leftarrow \sigma_{\ell-t}^2$
for $j \in [1 \ldots \ell - 2t]$ **do**
    No operation on σ_j
for $j \in [\ell - 2t \ldots \ell - t]$ **do**
    Set $\sigma_j = \sqrt{\sigma_j^2 - \delta_i}$
for $j \in [\ell - t \ldots \ell]$ **do**
    Set $\sigma_j = 0$

CHAPTER 4

EXPERIMENTS

Experiments were conduceted on real-world as well as synthetic datasets. The focus here is on tall and skinny matrices where the number of rows is far greater than the number of dimensions.

For synthetic datasets, we replicated the approach by [37]. Real-world datasets were chosen to capture the variation of the singular value distribution as much as possible, which is shown in the Figure 4.1.

4.1 Setup

We conducted experiments on two machines. The other machine has OpenSUSE 12.3 with 32 cores of Intel(R) Core(TM) i7-4770S CPU(3.10 GHz) and 32GB of RAM. The difference between the two machines is that the latter has SSD.

4.2 Datasets

It is important to look at a wide variety of dataset distributions for fair and satisfactory results. We consider four real-world datasets. QRPivot and ConnectUS are matrices taken from the University of Florida Sparse Matrix collection [1]. The singular values distribution has periodical drop-offs as seen in Figure 4.1, which makes it interesting when we project on a rank-k subspace and try to capture how much we have preserved the norms and subspace. The Birds dataset [2] consists of images of birds as rows and each column being a feature. The singular values distribution for Birds as seen in Figure 4.1 is almost flat. The Spam dataset construction is described in [34] and has a gradual concept drift. Concept drift is a phenomenon in which the properties of data change over time. We point the readers to [24,33,51] for more detail about concept drift and its types. The drop-off rate

in the singular values distribution as seen in Figure 4.1 is different compared to QRPivot and Birds.

For synthetic datasets with random noise, we replicate the datasets used in [37]. We generate an input matrix $n \times d$ matrix A where $A = SDU + F/\zeta$, SDU is the m-dimensional signal, and F/ζ is the (full) d-dimensional noise with ζ controlling the signal-to-noise ratio. The signal matrix $S \in \mathbb{R}^{n \times m}$ for each entry is set as $S_{i,j} \sim N(0,1)$ i.i.d. D is a diagonal matrix with entries $D_{i,i} = 1 - (i-1)/d$ linearly decreasing, and $U \in \mathbb{R}^{m \times d}$ is a random rotation. Entries of matrix F; $F_{i,j}$ are generated as $i.i.d.$ from a normal distribution $N(0,1)$, and we keep $\zeta = 10$ for the signal dimension $(20, 30)$. With signal dimension 30, we vary $\zeta = (30, 60)$ to see the impact of increasing noise. We also use the adversarial drift dataset used in [25]. Two orthogonal subspaces $S_1 = \mathbb{R}^{m_1}$ and $S_2 = \mathbb{R}^{m_2}$ ($m_1 = 400$ and $m_2 = 4$) are constructed on which we project two sets of random vectors and normalize them to construct the data matrix. The datapoints are ordered such that the first quarter of the dataset consists of vectors from the first subspace S_1, and the remaining portion of the dataset comprises vectors from S_2.

We summarize the datasets in the Table 4.1. The numeric rank is defined $\|A\|_F^2/\|A\|_2^2$ and the number of non-zero entries reflect dense and sparse datasets.

The singular values distribution of the datasets is given in Figure 4.1. In these plots, the x-axis represents the number of singular values and the y-axis shows the log of the singular values. We annotate the synthetic dataset Random Noisy, as $SDU20, SDU30$ where the signal dimension is $S = (20, 30)$ and the signal-to-noise ratio is $\zeta = 10$. We also vary the noise parameter for the $SDU30$ dataset where we set $\zeta = (30, 60)$ and these datasets, we denote as $SDU30\_30$ and $SDU30\_60$, respectively . We compute kurtosis for all the weights of the rows to determine if the rows follow a heavy-tailed distribution or light-tailed distribution. Kurtosis is a measure of the shape of the probability distribution of a given real-valued random variable. In Table 4.2, we compute the excess kurtosis. We follow Fisher's definition and the baseline kurtosis is 0, standardized for Normal Distribution. A positive excess kurtosis reflects fatter tails, whereas a negative excess kurtosis represents thinner tails and flatter distributions.

We can group the datasets based on their kurtosis values given in the Table 4.2 into three groups.

- QRPivot dataset having negative kurtosis has thinner tails compared to the rest of the datasets.
- Birds, CIFAR-10 and Random Noisy dataset have moderate heavy-tailed distributions.
- Spam, Connectus and Adversarial Drift datasets which have concept drift phenomenon all have high excess kurtosis and we see variation in results for this datasets primarily.

4.3 Evaluation Parameters

We look to answer the following questions in our experimental results:

- For a fixed rank k when computing the rank-k approximation, what are the best algorithms in terms of running time, covariance error, and projection error?
- Datasets can be sparse or dense following some distribution. What is the impact of such distributions on the running time and accuracy of the algorithms?
- For a desired error err, which algorithms are the fastest?
- Which algorithms give the best error, when running time is a constraint?
- Which algorithms give the best error, when space is a constraint?

All the randomized algorithms were run five times. We take the median value of error for all randomized algorithms.

4.4 APT

APT [48] is a platform for researchers to perform their experiments and make the experimental setup public for other researchers to access and build on it. This setup is for verifying and validating results published in research papers. We make our code, datasets, and plots available on APT for reproducibility.

Table 4.1: Datasets and properties about them.

DataSet	# Datapoints	# Attributes	Rank	Numeric Rank	NNZ%
CIFAR-10	60000	3072	3072	1.19	99.75
Connectus	394792	512	512	4.83	0.0055
QRPivot	660	749	660	98	0.0077
Birds	11789	312	312	12.50	100
Spam	9324	499	499	3.25	0.07
Random Noisy $(S, \zeta) = (20, 10)$	10000	500	500	12	100
Random Noisy $(S, \zeta) = (30, 10)$	10000	500	500	14.93	100
Random Noisy $(S, \zeta) = (30, 30)$	10000	500	500	11.16	100
Random Noisy $(S, \zeta) = (30, 30)$	10000	500	500	10.54	100
Adversarial	10000	500	500	1.69	100

Table 4.2: Kurtosis for all datasets

Dataset	# Excess Kurtosis
CIFAR-10	1.34
Connectus	17.60
QRPivot	-0.77
Birds	1.715
Spam	3.79
Random Noisy $(S, \zeta) = (20, 10)$	1.40
Random Noisy $(S, \zeta) = (30, 10)$	0.95
Random Noisy $(S, \zeta) = (30, 30)$	0.98
Random Noisy $(S, \zeta) = (30, 30)$	1.16
Adversarial	5.80

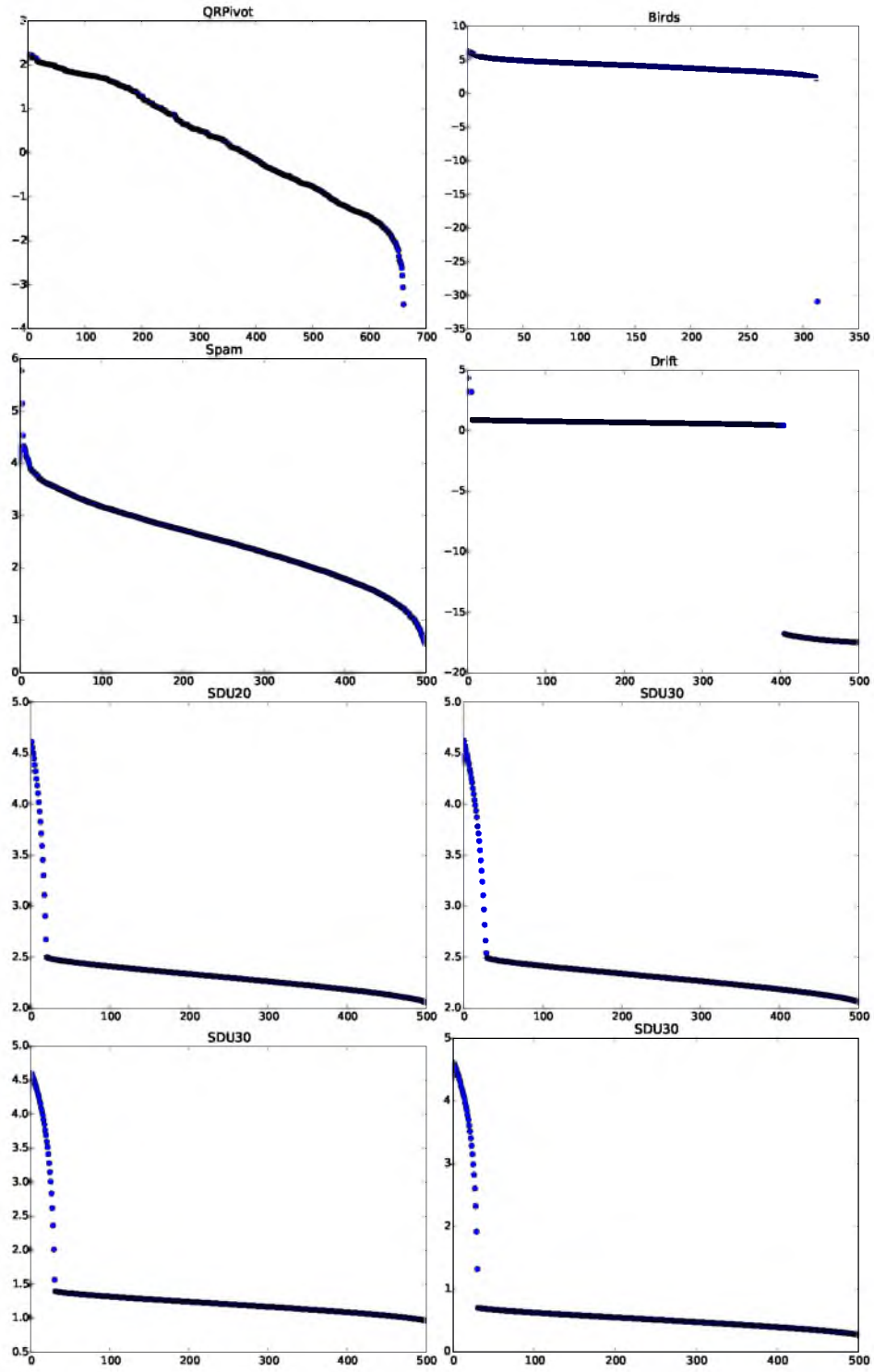

Figure 4.1. Singular Values Distribution: 1 : QRPivot, 2 : Birds, 3 : Spam, 4 : Adversarial Drift, 5 : Random Noisy $(S, \zeta) = (20, 10)$, 6 : Random Noisy with $(S, \zeta) = (30, 10)$, 7 : Random Noisy with $(S, \zeta) = (30, 30)$, 8 : Random Noisy with $(S, \zeta) = (30, 60)$.

CHAPTER 5

RESULTS

The results are organized as follows:

- Covariance Error

  $\mathsf{err} = \|A^T A - B^T B\|_2 / \|A\|_F^2$

- Projection Error 1: Projection onto k subspace and then get approximation

  $\mathsf{proj\text{-}err1} = \|A - \pi_{Q_k}(A)\|_F^2 / \|A - A_k\|_F^2$

- Projection Error 2: Projection onto subspace and then get k approximation

  $\mathsf{proj\text{-}err2} = \|A - (\pi_Q(A))_k\|_F^2 / \|A - A_k\|_F^2$

- Running Time

5.1 Dataset: Birds

5.1.1 Approximation Error vs Sketch Size

Sketch size is the number of rows maintained in the sketch matrix. We fix the rank $k = 10$ for all algorithms for the projection errors. The Birds dataset is a dense matrix with many nonzero entries. We keep the sketch size from $(20, \ldots, 100)$, which is very small compared to the actual rows of the matrix.

5.1.1.1 Covariance Error

Plots (a) and (b) in Figure 5.1 show results for FREQUENTDIRECTIONS and its variants and the tweaks we propose. In the first plot, we see 0.2 FREQUENTDIRECTIONS giving the best error among the algorithms based on FREQUENTDIRECTIONS, which is very close to the error achieved by ISVD. In the second plot, we show results for three additional algorithms. The first is Fast FREQUENTDIRECTIONS algorithm given by Edo

Libery in [37], and the other two are the tweaks proposed for FREQUENTDIRECTIONS and PARAMETERIZED FREQUENT DIRECTIONS. Both tweaks give errors close to the error given by $0.2-$FREQUENTDIRECTIONS and ISVD, whereas Fast FREQUENTDIRECTIONS has the highest error in this area of algorithms, but is faster, which we will see in the running time plots.

For Row/Column Sampling algorithms, we notice in Figure 5.1 that DETERMINISTIC LEVERAGE SCORE SAMPLING has the highest error over all sketch sizes. This is likely due to lack of rescaling done in the rows/columns selected in DETERMINISTIC LEVERAGE SCORE SAMPLING. Reservoir sampling algorithms PRIORITY SAMPLING, WEIGHTED RESERVOIR SAMPLING W/O REPLACEMENT, WEIGHTED RESERVOIR SAMPLING WITH REPLACEMENT, and VARIANCE OPTIMAL SAMPLING do not perform as well as LINEAR TIME SVD for sketch size $\ell = (20, 50)$, but converge better as sketch size increases.

As we see in the kurtosis Table 4.2, the Birds dataset is relatively light tailed compared to datasets such as Spam and ConnectUS. When performing one of the reservoir sampling algorithms such as PRIORITY SAMPLING, there might be situations where we draw random numbers that favor the lower-weighted items and very small random numbers for the heavy to moderately weighted rows. The reservoir sampling techniques would pick a lower-moderately weighted row instead of heavy rows, which impacts the threshold over the whole stream and at the end impacts the rescaling estimate.

For the random-projections based algorithms in Figure 5.1 (c), we see the sparse constructions have errors close to CW TRANSFORM - 09 algorithm, which is very good for dense matrices and also give respectable accuracy for other datasets too. FAST JLT gives the best error in this class, but is not fast as the sparse constructions. TOEPLITZ and CZ TRANSFORM - 14 do not perform well as the appropriate normalization constants are not known for them when computing covariance matrices in the Spectral norm.

5.1.1.2 Projection Error

Most of the algorithms are able to preserve subspace as shown in the Figure 5.2, but there is clear gap between FREQUENTDIRECTIONS variants and other algorithms. PRIORITY SAMPLING, VARIANCE OPTIMAL SAMPLING, WEIGHTED RESERVOIR SAMPLING WITH REPLACEMENT, and WEIGHTED RESERVOIR SAMPLING W/O REPLACEMENT perform better than the algorithms LINEAR TIME SVD and SUBSPACE SAMPLING, but match

DETERMINISTIC LEVERAGE SCORE SAMPLING. TOEPLITZ, CW TRANSFORM - 13, and OSNAP are able to do as well as the dense random projection algorithms.

5.1.2 Running Time vs Sketch Size

All the algorithms besides SUBSPACE SAMPLING and DETERMINISTIC LEVERAGE SCORE SAMPLING are impelemented in streaming model with memory constrained to process only a row at a given time and have rest of the memory for the sketch matrix. In plot(b) in Figure 5.3, we show the running time improvements achieved by the tweaks and Fast FREQUENTDIRECTIONS over ISVD and 0.2FREQUENTDIRECTIONS. The reservoir sampling algorithms we introduced are the fastest in sampling algorithms, and are streaming algorithms. Notice the sampling with replacement algorithms WEIGHTED RESERVOIR SAMPLING WITH REPLACEMENT and LINEAR TIME SVD running time increases polynomially as the sketch size increases. We make this more evident in the next section. TOEPLITZ is faster than the dense RANDOM PROJECTION algorithms and the sparse constructions CW TRANSFORM - 13, OSNAP, and CZ TRANSFORM - 14 are the fastest algorithms among all three areas.

5.1.3 Leading algorithms

FREQUENTDIRECTIONS variants give the best approximation error and hence the sketch among all the areas as shown in Figures 5.4 and 5.5.

The plot in Figure 5.6(a) shows all the leading algorithms where 0.2 FD is the slowest algorithm, while the improvement Tweak 0.2 FD is much more competitive with the rest of the algorithms. Plot in Figure 5.6(b) shows PRIORITY SAMPLING being faster than LINEAR TIME SVD, and hence it is the fastest sampling algorithm for this dataset. We will see later that VARIANCE OPTIMAL SAMPLING is the faster algorithm as sketch size increases.

5.2 Dataset: Spam

5.2.1 Approximation Error vs Sketch Size

The Spam dataset is a sparse matrix but is tall and skinny, too. The Spam dataset is a prototypical text dataset, with some of the vectors having very heavy mass besides being sparse.

5.2.1.1 Covariance Error

We find our reservoir sampling algorithms performing much better than LINEAR TIME SVD, SUBSPACE SAMPLING, DETERMINISTIC LEVERAGE SCORE SAMPLING as shown in plot (b) in Figure 5.7. This is a scenario where importance sampling possibly does slightly worse. If there are very few heavy items that come before numerous smaller items, then the probability of choosing the smaller items is very small, and several such items not being picked leads to no rescaling and hence not preserving distances optimally. Again, FREQUENTDIRECTIONS and its variants outperform all the other algorithms.

5.2.1.2 Projection Error

For the projection error, DETERMINISTIC LEVERAGE SCORE SAMPLING and SUBSPACE SAMPLING perform poorly. This performance is due to the leverage scores finding the most important directions, but missing out on the less important, but a numerous number of smaller directions. As the sketch size increases, the error starts converging as shown in Figure 5.8. This showcases one problem where DETERMINISTIC LEVERAGE SCORE SAMPLING and SUBSPACE SAMPLING might need more samples than other algorithms to have good appproximation errors.

5.2.2 Leading algorithms for Dataset: Spam

FREQUENTDIRECTIONS variants give the best approximation error and hence the sketch among all the algorithm areas, as shown in the Figures 5.9 and 5.10.

5.3 Approximation Error vs Running Time

5.3.1 Birds

In the previous section we saw FREQUENTDIRECTIONS and its variants having the best approximation error but are slow in comparison to sampling and random projection algorithms. We are interested in finding, given the error threshold err by FREQUENTDIRECTIONS and its variants for the smallest sketch size ℓ, how the algorithms in sampling and random projections perform for running time and the sketch size for obtaining error err. For the Birds dataset, we take the error threshold err, given by Tweak 0.2 FREQUENTDIRECTIONS for sketch size $\ell = 20$.

As shown in Figures 5.11 and 5.12, SUBSPACE SAMPLING, LINEAR TIME SVD, and

CW TRANSFORM - 09 are able to get close to the error threshold err, but the running time is in the same range as Tweak 0.2 FREQUENTDIRECTIONS and increases to attain err. In addition, the sketch size ℓ is significantly more as shown in Figure 5.13. On the other hand, CW TRANSFORM - 13, OSNAP, PRIORITY SAMPLING, VARIANCE OPTIMAL SAMPLING are able to achieve err much more quickly again with a high sketch size ℓ as seen in Figure 5.13. We note here that VARIANCE OPTIMAL SAMPLING is faster than PRIORITY SAMPLING. This is due to the rescaling procedures in the given algorithms. In PRIORITY SAMPLING, we go over all the items and rescale them as stated in the last steps of the algorithm 3.1.6. On the other hand, in VARIANCE OPTIMAL SAMPLING, we rescale only items in list T, which is a very tiny fraction of the overall number of items in the sketch. Hence as sketch size increases, PRIORITY SAMPLING running time increases to some extent compared to VARIANCE OPTIMAL SAMPLING. Fast FREQUENTDIRECTIONS is able to get the err for a much smaller sketch size compared to the rest of the algorithms. Hence, if running time is a constraint with no constraints on space, CW TRANSFORM - 13, OSNAP, PRIORITY SAMPLING, and VARIANCE OPTIMAL SAMPLING are the best alternatives, whereas if space is a constraint, then Tweak 0.2 FREQUENTDIRECTIONS and Fast FREQUENTDIRECTIONS are the best alternative. We present the sketch size ℓ used for the leading algorithms experiments in the Table 5.1.

5.3.2 CIFAR-10

CIFAR-10 is a benchmark dataset in Computer Vision. It consists of 60000 images where each image has a RGB component of $3 \times 32 \times 32$, hence giving the raw dataset with 60000 rows and 3072 columns; it is a dense matrix.

We look to answer the same question, given the error threshold err by FREQUENTDIRECTIONS and its variants for the smallest sketch size ℓ: How do the algorithms in sampling and random projections perform for running time and the sketch size for getting error err?

For the given dataset, we show the covariance error as the projection error follows similar trends.

In Figure 5.14, we give the overview of the leading algorithms where algorithms SUBSPACE SAMPLING, DETERMINISTIC LEVERAGE SCORE SAMPLING, and CW TRANSFORM - 09 running time becomes prohibitive to attain error err. In Figure 5.15, we

look at close-up plot where we see Fast FREQUENTDIRECTIONS matching Tweak 0.2 FREQUENTDIRECTIONS for the error err and the running time, but the sketch size for Fast FREQUENTDIRECTIONS is slightly more than the Tweak 0.2− FREQUENTDIRECTIONS. Also notice the running time growth of LINEAR TIME SVD with the increase in sketch size. In Figure 5.16 we look more closely at the performance of VARIANCE OPTIMAL SAMPLING, PRIORITY SAMPLING, CW TRANSFORM - 13, and OSNAP. We notice VARIANCE OPTIMAL SAMPLING does better than all the other algorithms for both running time and error; though the sketch size is much more than both Tweak - 0.2 FREQUENTDIRECTIONS and Fast FREQUENTDIRECTIONS, as seen in Figure 5.17. Notice that for this given dense matrix, CW TRANSFORM - 13 and OSNAP perform slightly worse compared to their performance over sparse datasets and throughout. We present the sketch size ℓ used for the experiments in the Table 5.2.

5.3.3 ConnectUS

ConnectUS is sparse matrix representing a recommendation system. The rows correspond to the web pages and columns correspond to unqiue users. The entries of the matrix are either 0 or 1, where 1 represents if a user favored a web page and 0 for no opinion about the page. It is very sparse similar to spam and has 394792 rows and 512 columns.

In Figure 5.18, we show the approximation error in comparison to the running time. All algorithms except DETERMINISTIC LEVERAGE SCORE SAMPLING give the error threshold given by Tweak 0.2− FREQUENTDIRECTIONS. Noticeably, the running time of LINEAR TIME SVD, SUBSPACE SAMPLING, and CW TRANSFORM - 09 is much higher than the Tweak 0.2− FREQUENTDIRECTIONS algorithm. We look at other algorithms in more detail in Figure 5.19 below. Since this is a sparse matrix, CW TRANSFORM - 13 and OSNAP give the best running time and are much faster than the fastest sampling algorithm VARIANCE OPTIMAL SAMPLING and PRIORITY SAMPLING. Even though CW TRANSFORM - 13 and OSNAP are faster to achieve the best error, they take two times more space than VARIANCE OPTIMAL SAMPLING and PRIORITY SAMPLING, as shown in Figure 5.20.

5.4 Other Results

We present our results for other datasets, and discuss a few changes that we see, but overall the trend remains the same.

5.4.1 Dataset: SDU20 and SDU30

All the algorithms follow the behavior as expected on the synthetic datasets. Notice that DETERMINISTIC LEVERAGE SCORE SAMPLING performs very well on the projection error in Figure 5.24 and is very close to FREQUENTDIRECTIONS variants for the error accuracy as sketch size increases and is also faster than FREQUENTDIRECTIONS. The most noticeable algorithm is Tweak-FREQUENTDIRECTIONS that we proposed, which performs significantly worse and demonstrates the lack of theoretical guarantees. In addition, the performance of hashing algorithms, CW TRANSFORM - 13, OSNAP, and CZ TRANSFORM - 14, is noticeably worse for all the synthetic datasets.

5.4.1.1 Covariance Error

Covariance error plots are shown in Figures 5.21 and 5.22.

5.4.1.2 Projection Error: 1

Projection error plots are shown in Figures 5.23 and 5.24.

5.4.1.3 Projection Error: 2

Projection error plots are shown in Figures 5.25 and 5.26.

5.4.2 Dataset: SDU30_30, SDU30_60

For this set of datasets, where we vary the signal-to-noise ratio, we expect the algorithms to show the same trends as the above synthetic datasets, but the error value to be worse, which is evident in the plots.

5.4.2.1 Covariance Error

Covariance error plots are shown in Figures 5.27 and 5.28.

5.4.2.2 Projection Error: 1

We show plots only for projection error - 1 in Figures 5.29 and 5.30, as projection error - 2 follows similar trends.

5.4.3 Dataset : QRPivot

QRPivot, besides being a sparse matrix, has very few heavy items, which are captued in the sketch given by DETERMINISTIC LEVERAGE SCORE SAMPLING. Hence it performs well on the covariance error measure, but the trend is flat, which is similar to other datasets. Lack of rescaling is a natural weakness of DETERMINISTIC LEVERAGE SCORE SAMPLING algorithms which leads to weaker covariance error, but is relatively stronger for QRPivot.

5.4.3.1 Covariance Error

Covariance error is shown in Figure 5.31

5.4.3.2 Projection Error

Projection error is shown in Figure 5.32

5.4.3.3 Leading algorithms

Leading algorithms for QRPivot for covariance error shown in Figures 5.33 and 5.34.

5.4.4 Dataset: Adversarial Drift

For this dataset, DETERMINISTIC LEVERAGE SCORE SAMPLING performs significantly worse than all the algorithms for approximation accuracy. Recollect that the adversarial drift dataset is made up of datapoints from two orthogonal subspaces. The highest leverage scores correspond to datapoints from both subspaces. Interestingly, the majority of the datapoints are from the less dominant subspace, and hence the projection error is very weak. Sampling more rows/columns will fix the situation but showcases the unreliable nature of DETERMINISTIC LEVERAGE SCORE SAMPLING. This can also be improved by using a PRIORITY SAMPLING or VARIANCE OPTIMAL SAMPLING step over the leverage scores to choose rows/columns, proving this theoretically though is an open question.

5.4.4.1 Covariance Error

Covariance error is shown in Figure 5.35

5.4.4.2 Projection Error

Projection error is shown in Figure 5.36

5.4.4.3 Leading algorithms

Leading algorithms for Adversarial Drift for covariance error shown in Figure 5.37 and 5.38.

5.5 Application: Lena

One of the applications of low-rank approximation is image compression as well as capturing the best features of the image with a smaller rank. It is possible that pictures taken with a camera have noise in them due to sensors associated with camera, and most the smaller singular values do not really give any meaningful data. Discarding such singular values is akin to having a low-rank approximation. We ran our leading algorithms on test image Lena. The image dimensions are 512×512. We construct a sketch matrix with 100 samples and then compute the low-rank approximation by projecting it on a rank 50 subspace. The rank of the matrix is 507 and we reduce its rank to 50 with the help of 100 samples. The leading algorithms low rank output is given in Figure 5.39

FREQUENTDIRECTIONS variants give the best image for the rank 50. Most of the algorithms produce good images including PRIORITY SAMPLING and TOEPLITZ. LINEAR TIME SVD which gives additive error guarantees, produces a low quality image. DETERMINISTIC LEVERAGE SCORE SAMPLING and SUBSPACE SAMPLING give decent images but not better than FREQUENTDIRECTIONS and its variants .

Table 5.1: Birds: Sketch size for leading algorithms

Algorithm	# Sketch Size for error threshold
Priority Sampling	(20, 500, 1000, 2000)
Deterministic Leverage Scores	(20, 500, 1000, 2000)
Subspace Sampling	(20, 1000, 2000, 3000)
LinearTime SVD	(20, 1000, 2000, 3000)
CW09	(20, 500, 1000, 2000)
OSNAP	(20, 500, 1000, 2000)
Fast FD	(20, 60, 100)
Tweak - 0.2 FD	(20, 60, 100)
CW13	(20, 500, 1000, 2000)

Table 5.2: CIFAR-10: Sketch size for leading algorithms

Algorithm	# Sketch size for error threshold
Priority Sampling	(20, 1000, 5000)
Deterministic Leverage Scores	(20, 1000, 5000)
Subspace Sampling	(20, 1000, 10000)
LinearTime SVD	(20, 1000, 10000)
CW09	(20, 1000, 5000)
OSNAP	(20, 1000, 10000)
Fast FD	(20, 60, 100)
Tweak - 0.2 FD	(20)
CW13	(20, 1000, 10000)

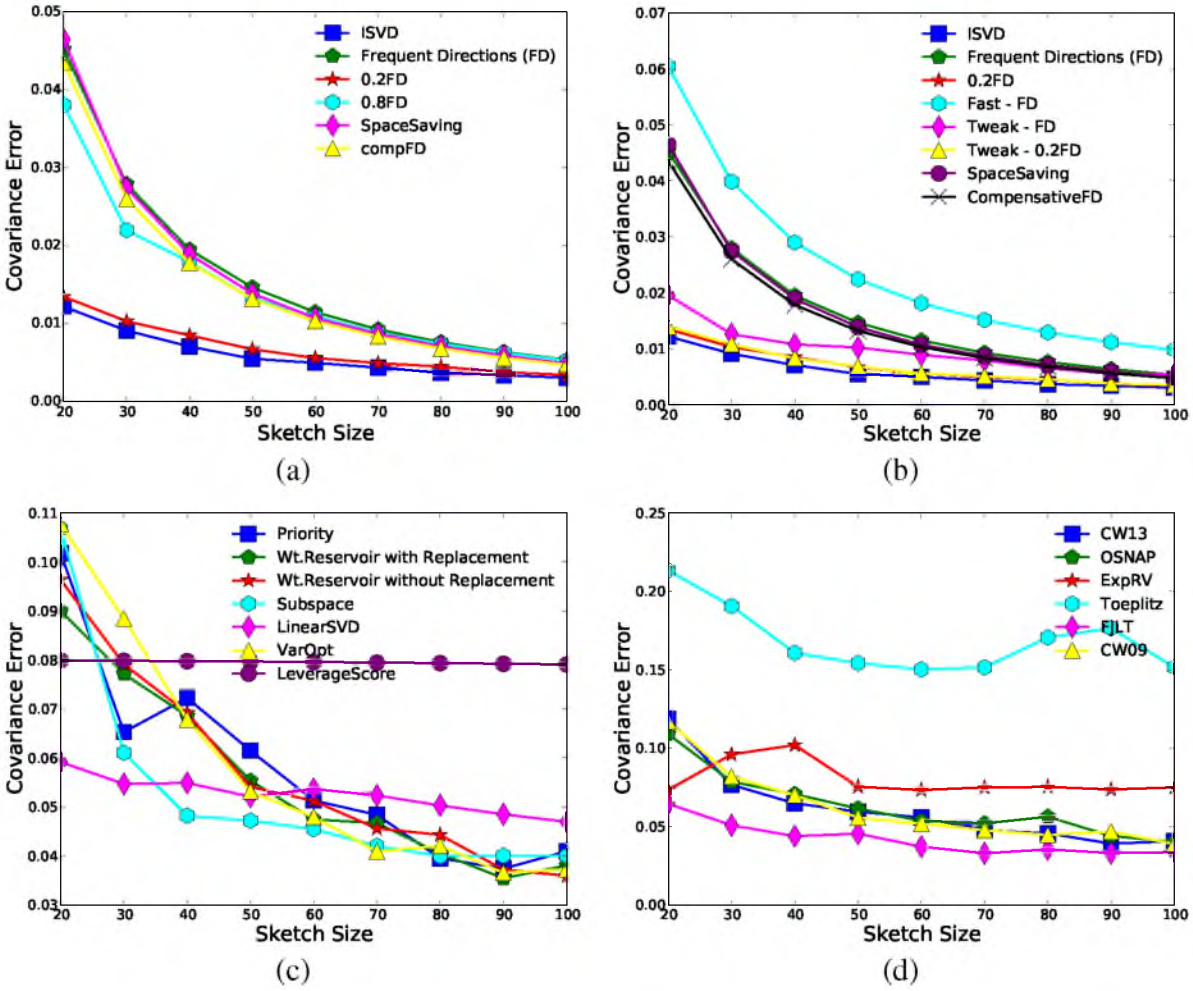

Figure 5.1. Birds: Covariance error, a) Algorithms in Frequent Directions area, b) Tweaks to algorithms in Frequent Direction area, c) Algorithms in Column Sampling area, d) Algorithms in Random Projections area

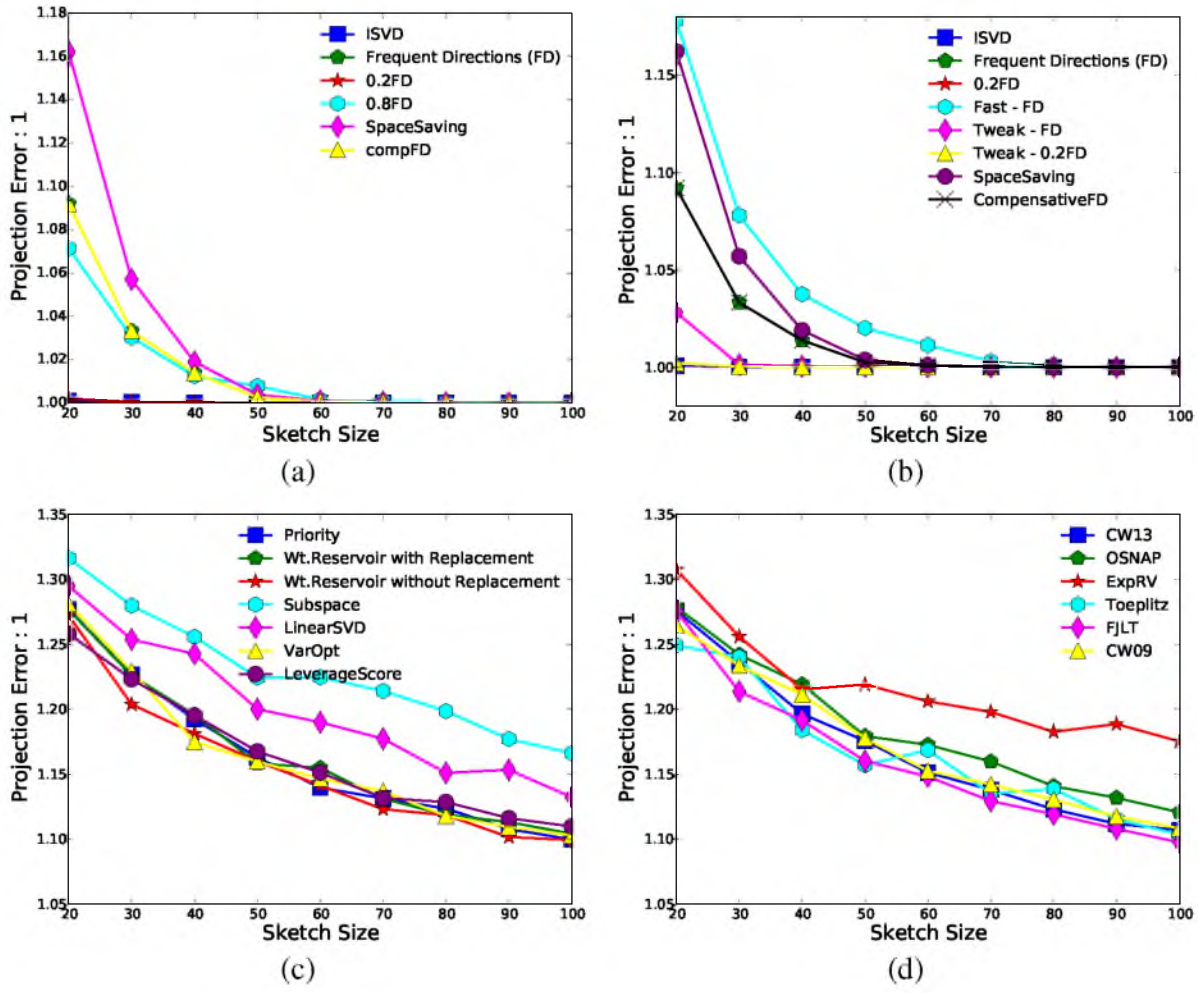

Figure 5.2. Birds: Projection error, a) Algorithms in Frequent Directions area, b) Tweaks to algorithms in Frequent Direction area, c) Algorithms in Column Sampling area, d) Algorithms in Random Projections area

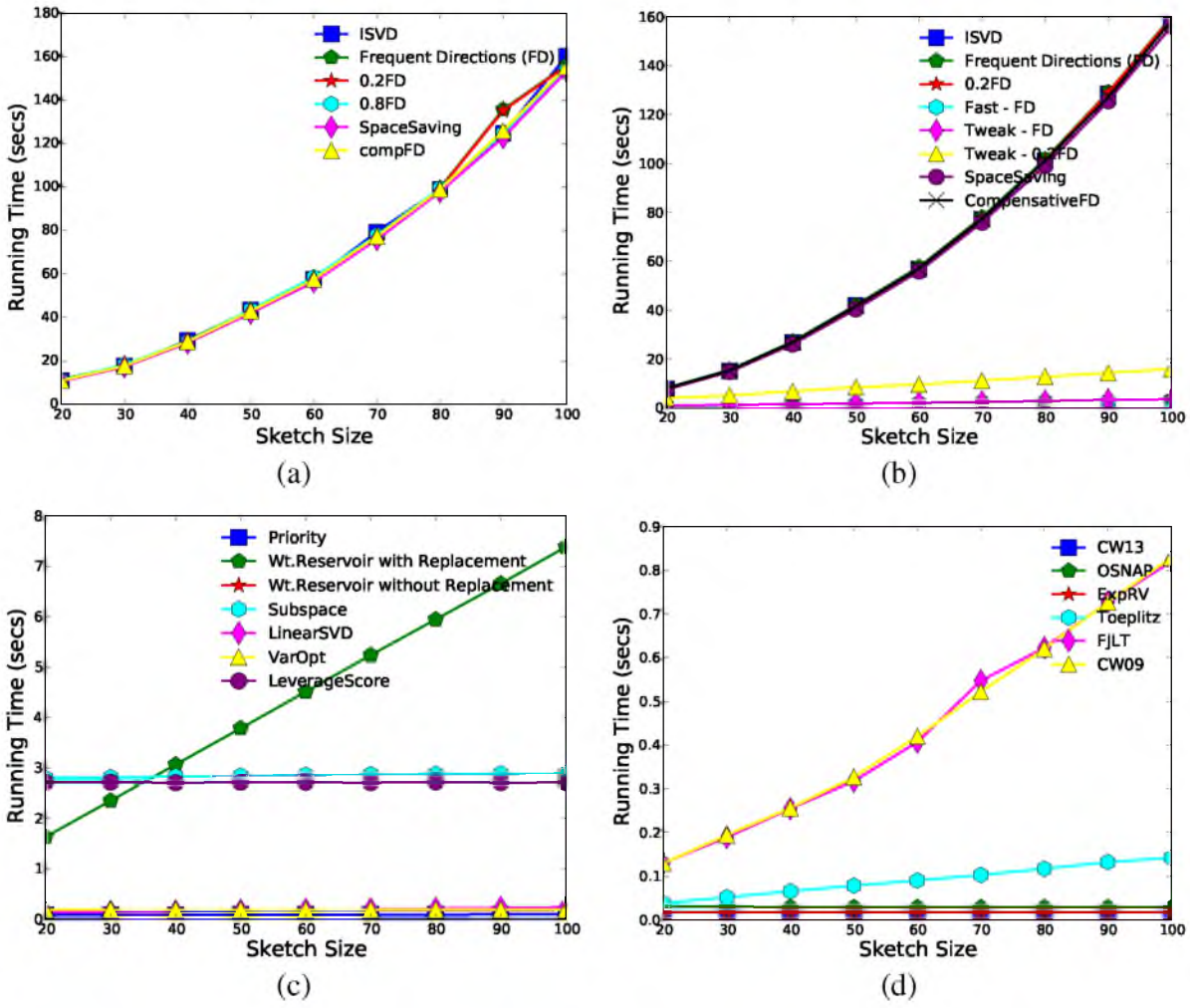

Figure 5.3. Birds: Running time, a) Algorithms in Frequent Directions area, b) Tweaks to algorithms in Frequent Direction area, c) Algorithms in Column Sampling area, d) Algorithms in Random Projections area

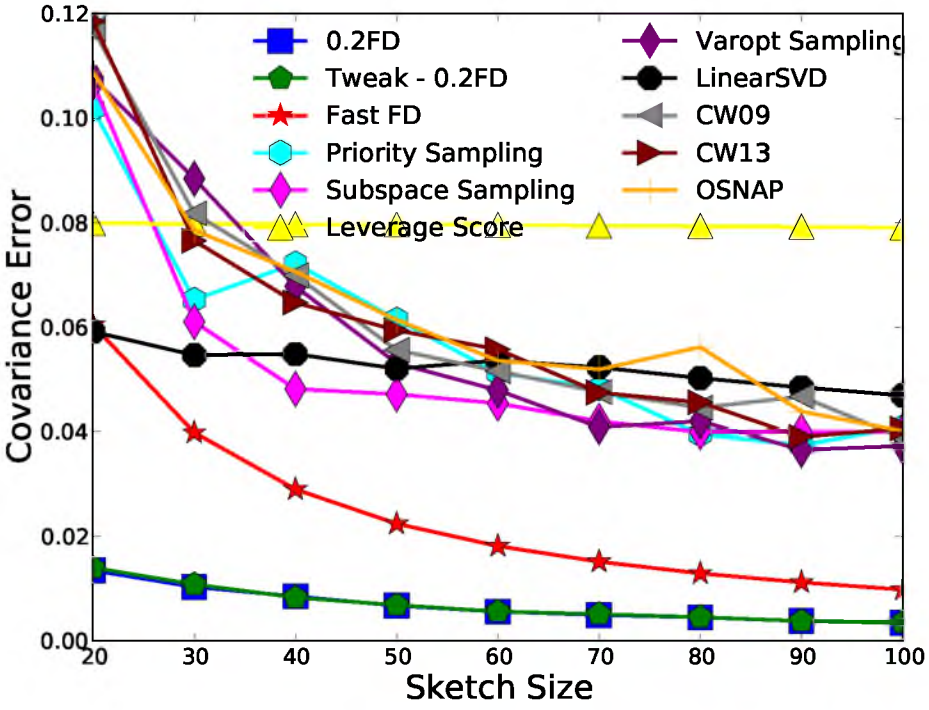

Figure 5.4. Leading algorithms for Birds: Covariance error

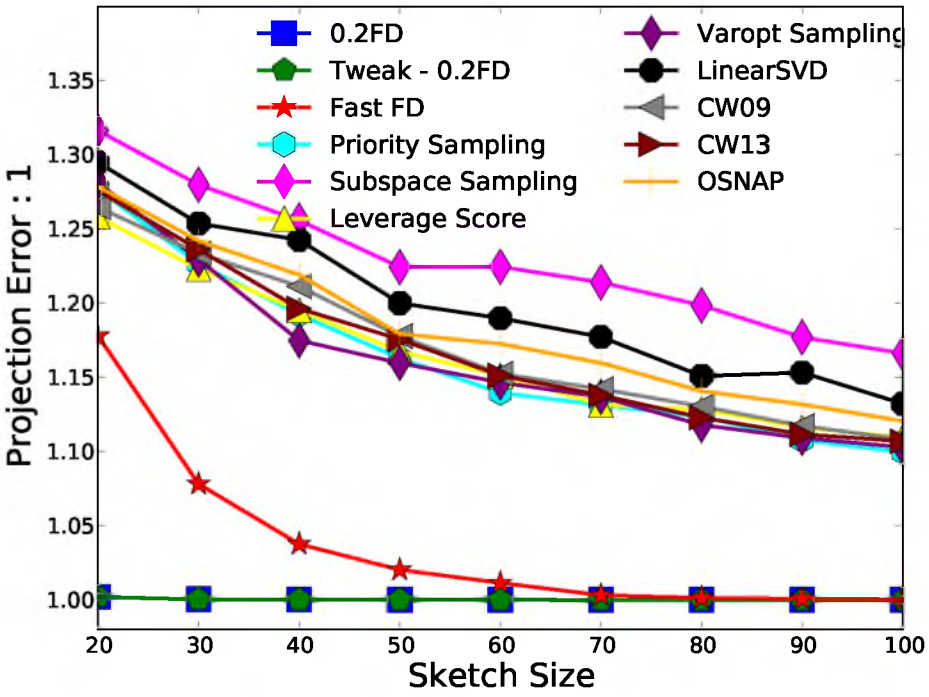

Figure 5.5. Leading algorithms for Birds: Projection error

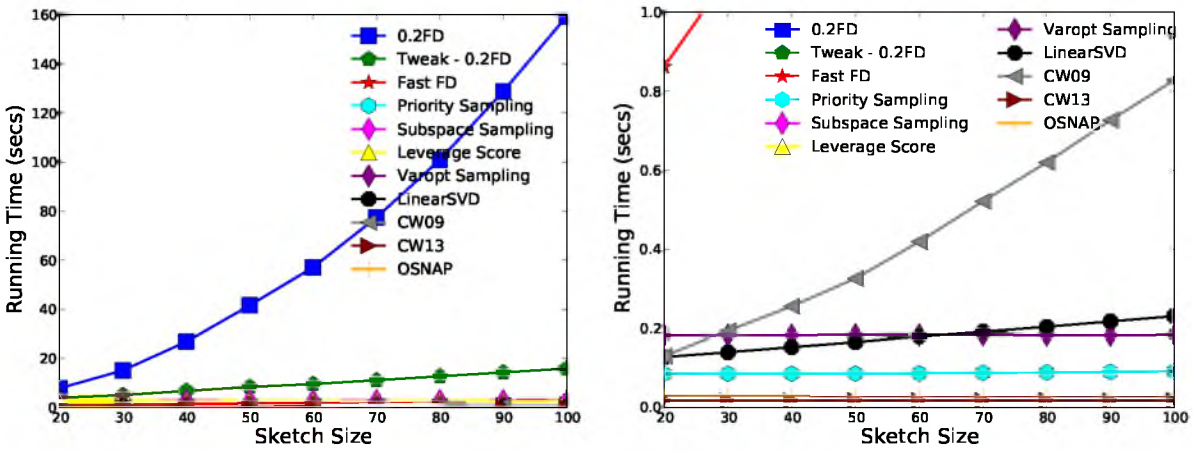

Figure 5.6. Birds: Running time, a) Overview plot for all leading algorithms, b) Close-up plot to show important differences

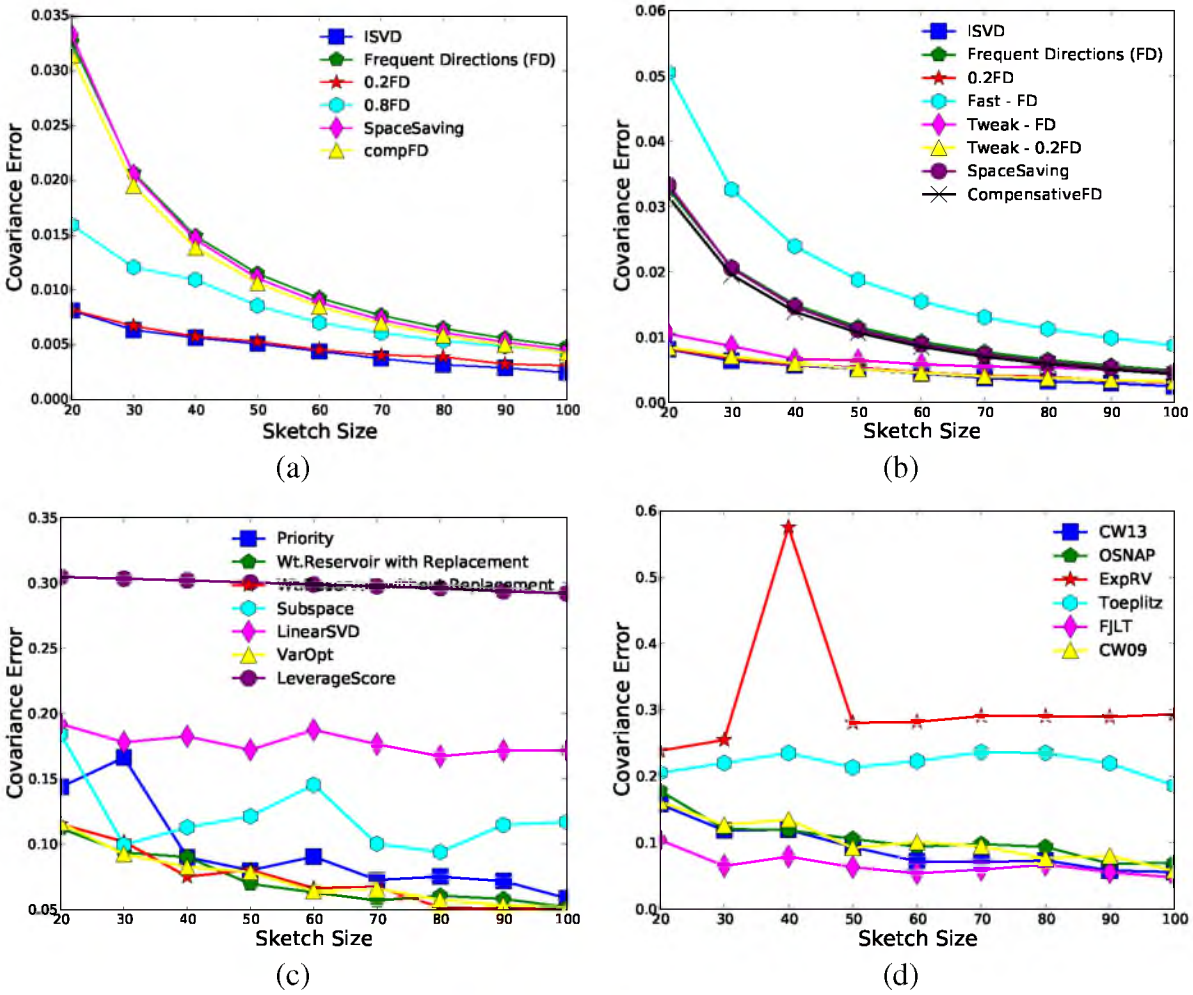

Figure 5.7. Spam: Covariance error, a) Algorithms in Frequent Directions area, b) Tweaks to algorithms in Frequent Direction area, c) Algorithms in Column Sampling area, d) Algorithms in Random Projections area

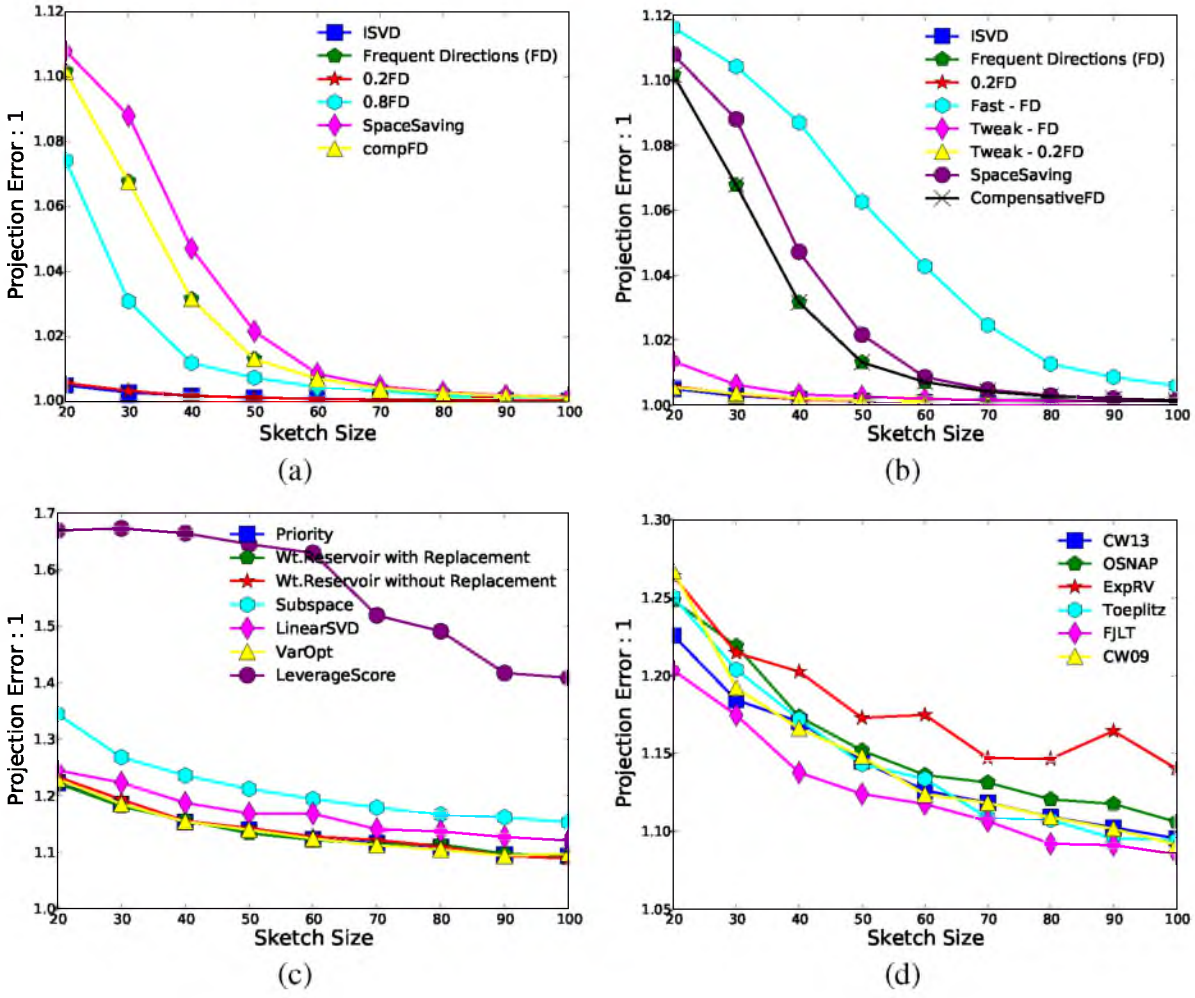

Figure 5.8. Spam: Projection error, a) Algorithms in Frequent Directions area, b) Tweaks to algorithms in Frequent Direction area, c) Algorithms in Column Sampling area, d) Algorithms in Random Projections area

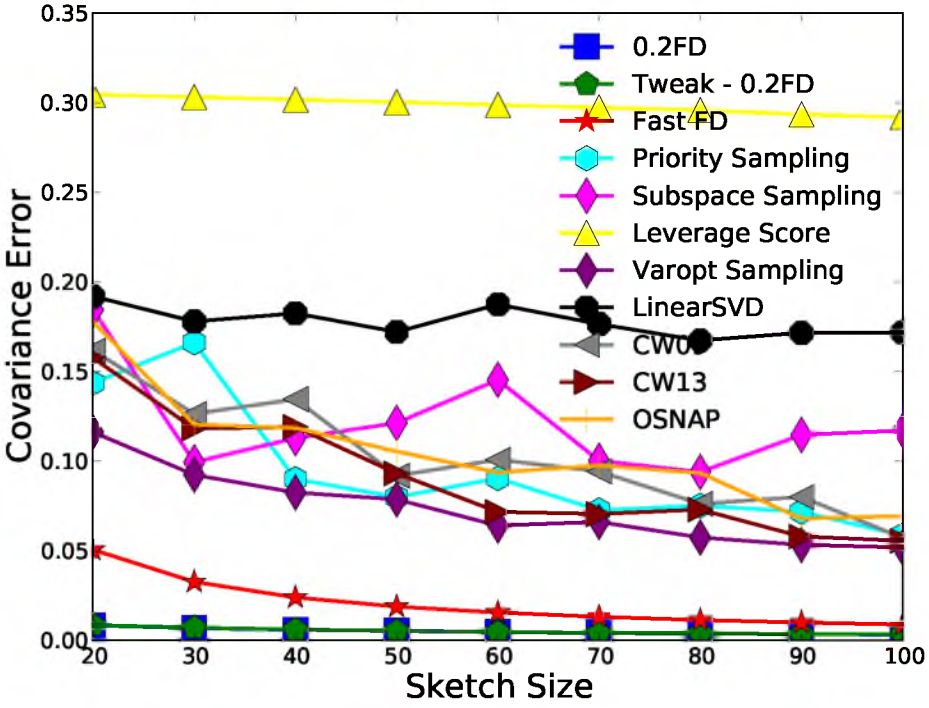

Figure 5.9. Leading algorithms for Spam: Covariance error

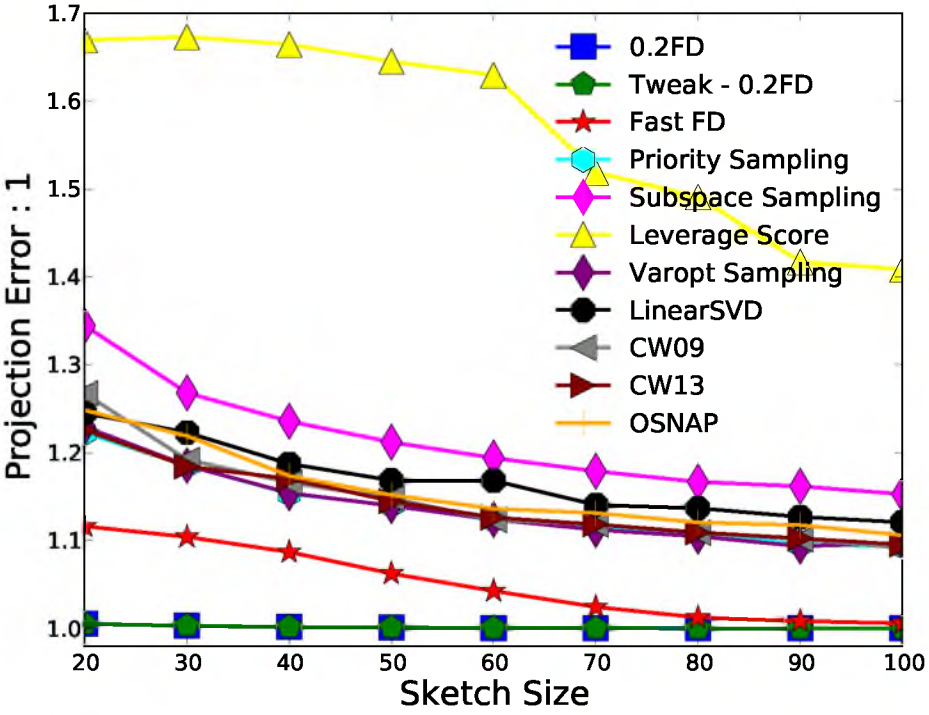

Figure 5.10. Leading algorithms for Spam: Projection error

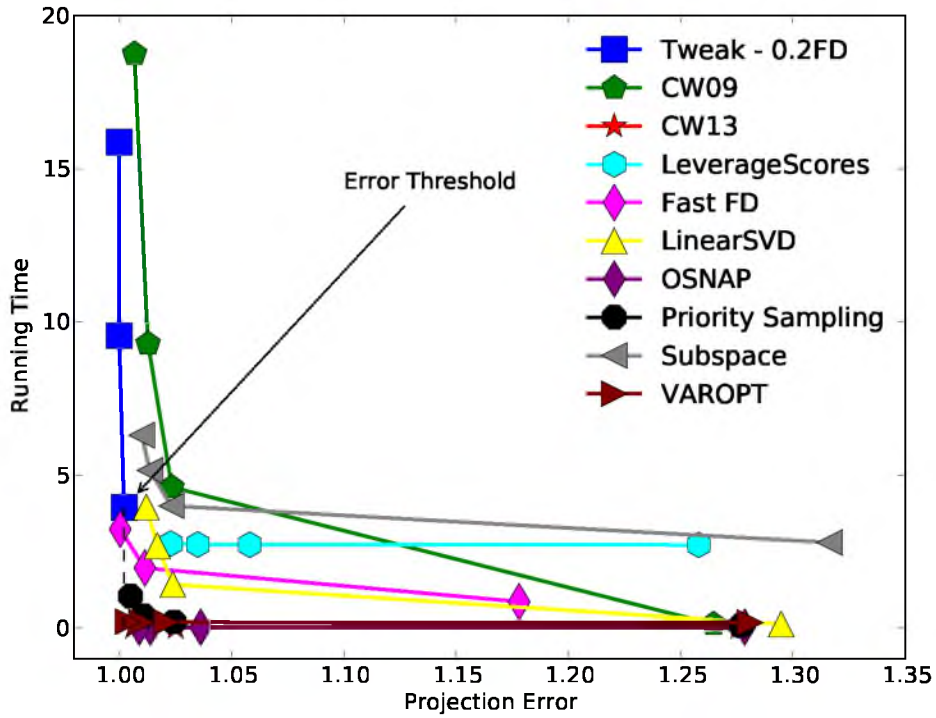

Figure 5.11. Error vs Running time leading algorithms for Birds: overview

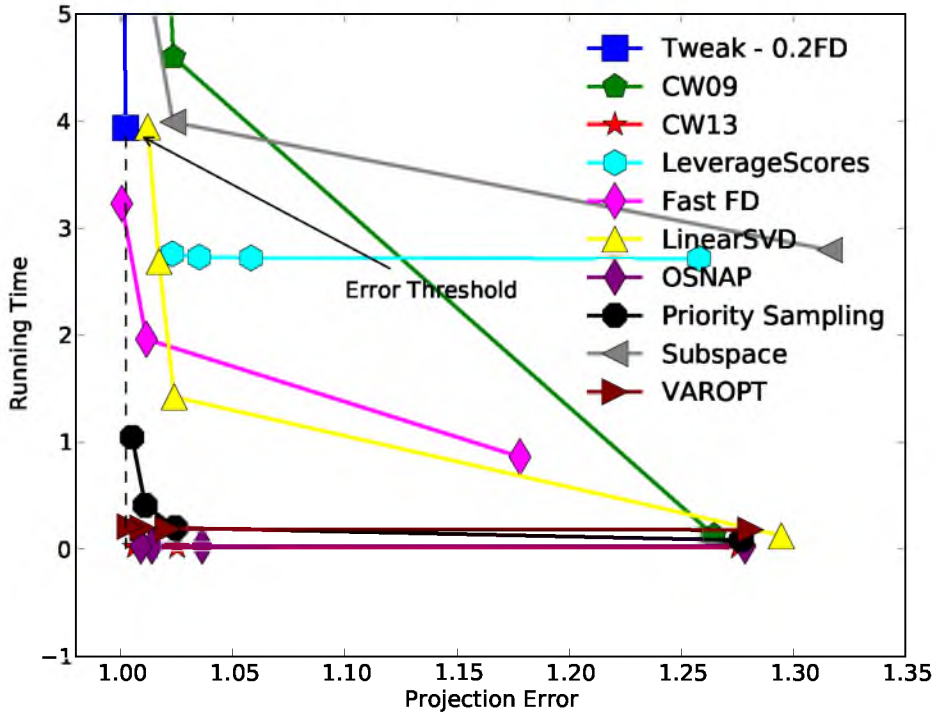

Figure 5.12. Error vs Running time leading algorithms for Birds: closeup

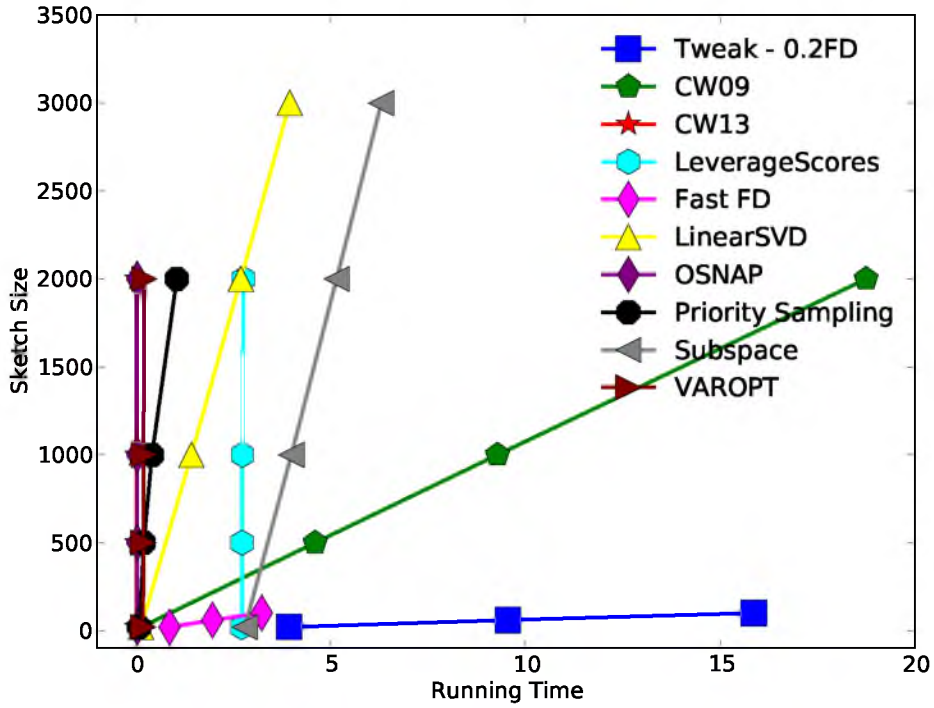

Figure 5.13. Running time vs sketch size for Birds

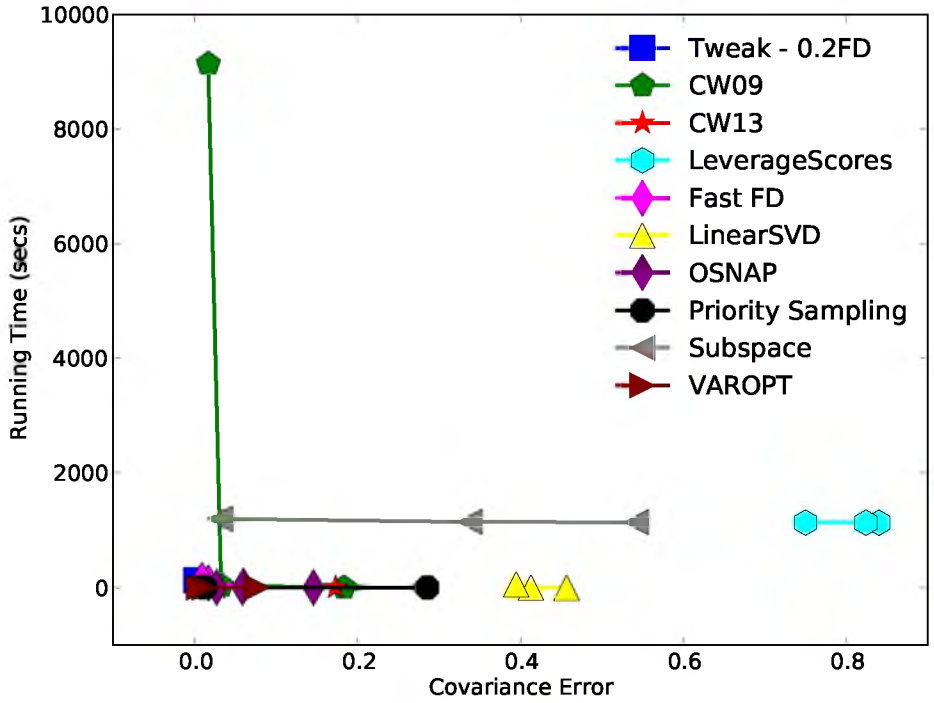

Figure 5.14. Error vs Running time leading algorithms for CIFAR-10: overview

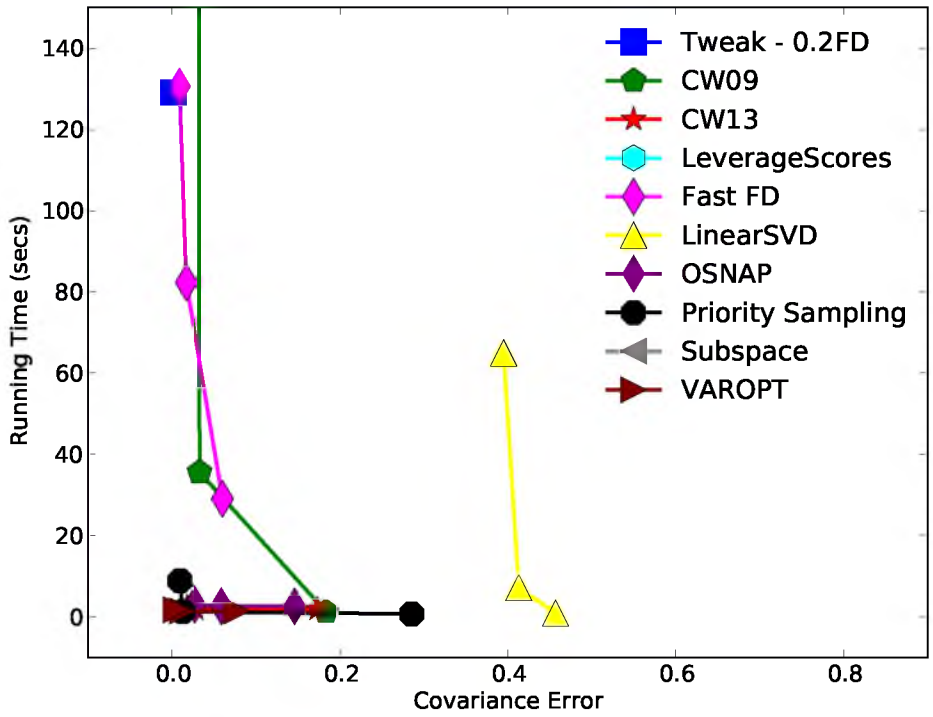

Figure 5.15. Error vs Running time leading algorithms for CIFAR-10: close-up1

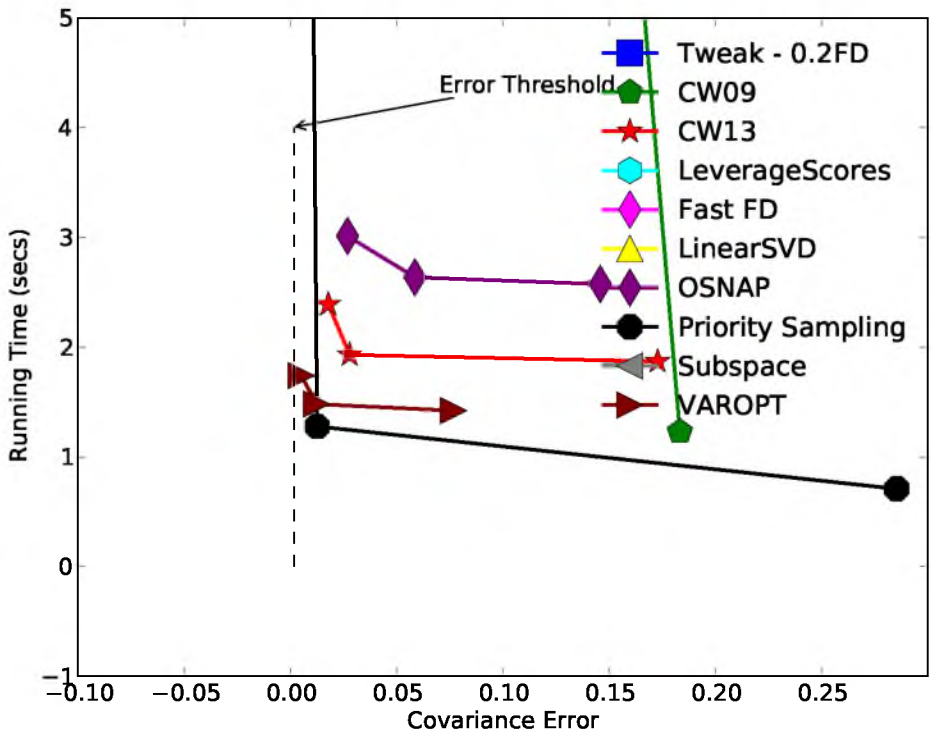

Figure 5.16. Error vs Running time leading algorithms for CIFAR-10: close-up2

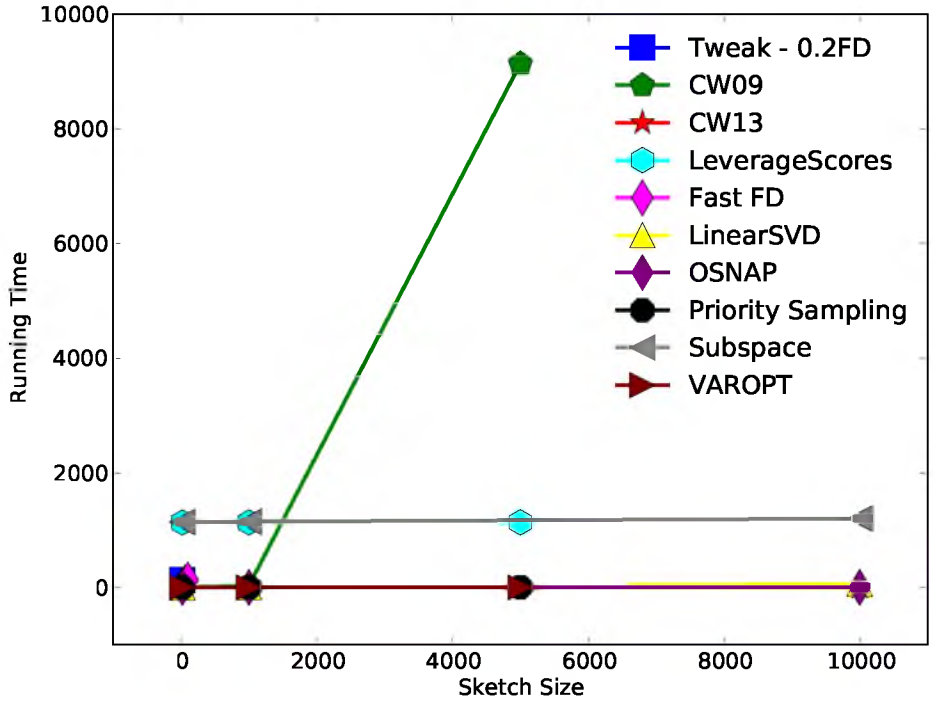

Figure 5.17. Sketch Size vs Running time for CIFAR-10

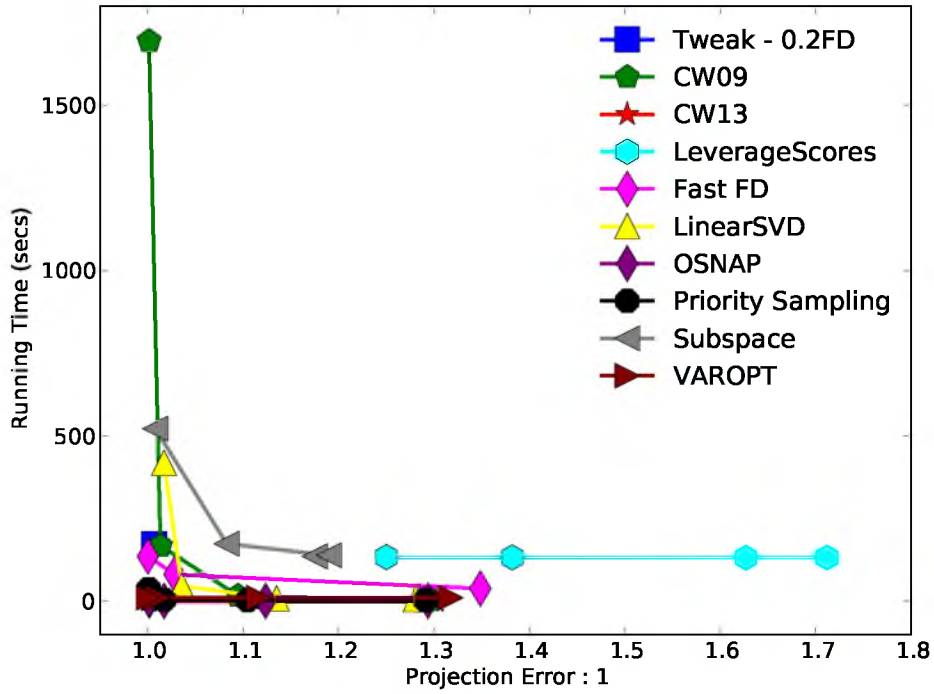

Figure 5.18. Error vs Running time leading algorithms for ConnectUS: overview

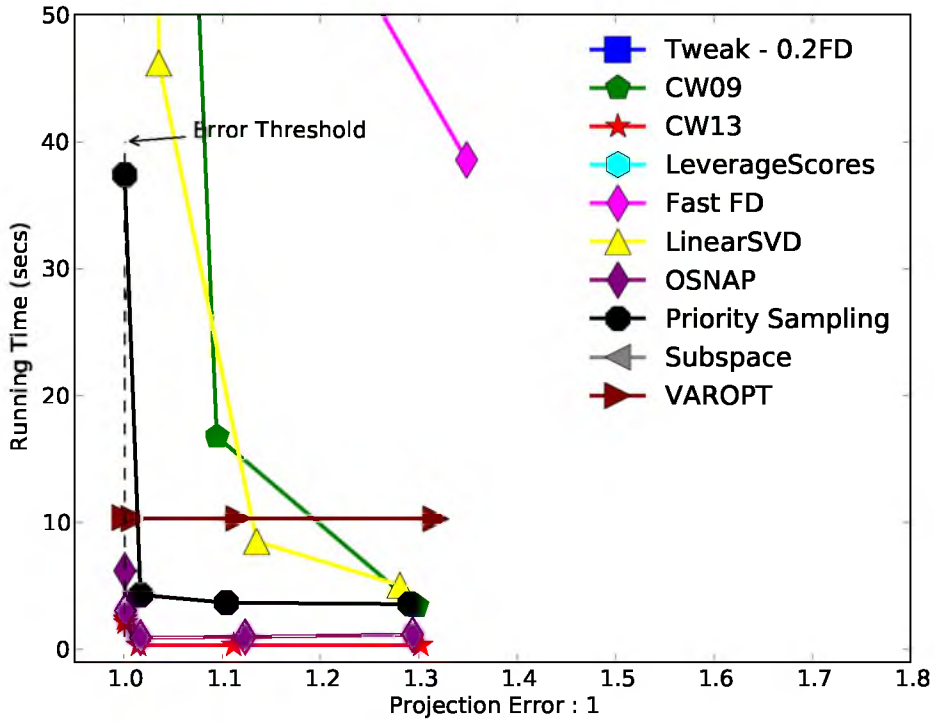

Figure 5.19. Error vs Running time leading algorithms for ConnectUS: close-up

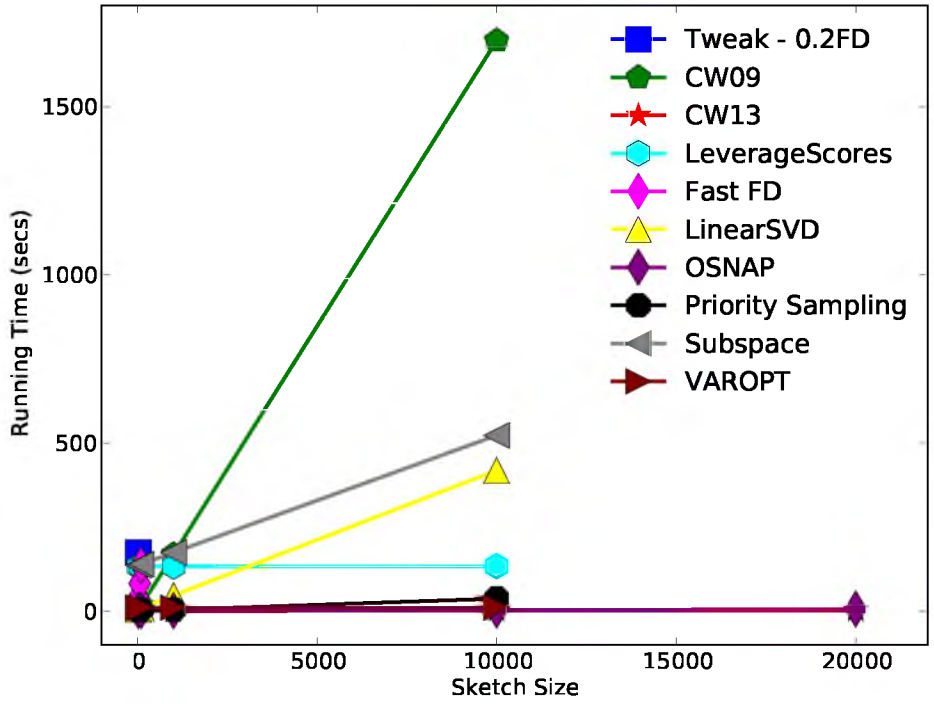

Figure 5.20. Sketch size vs Running time for ConnectUS

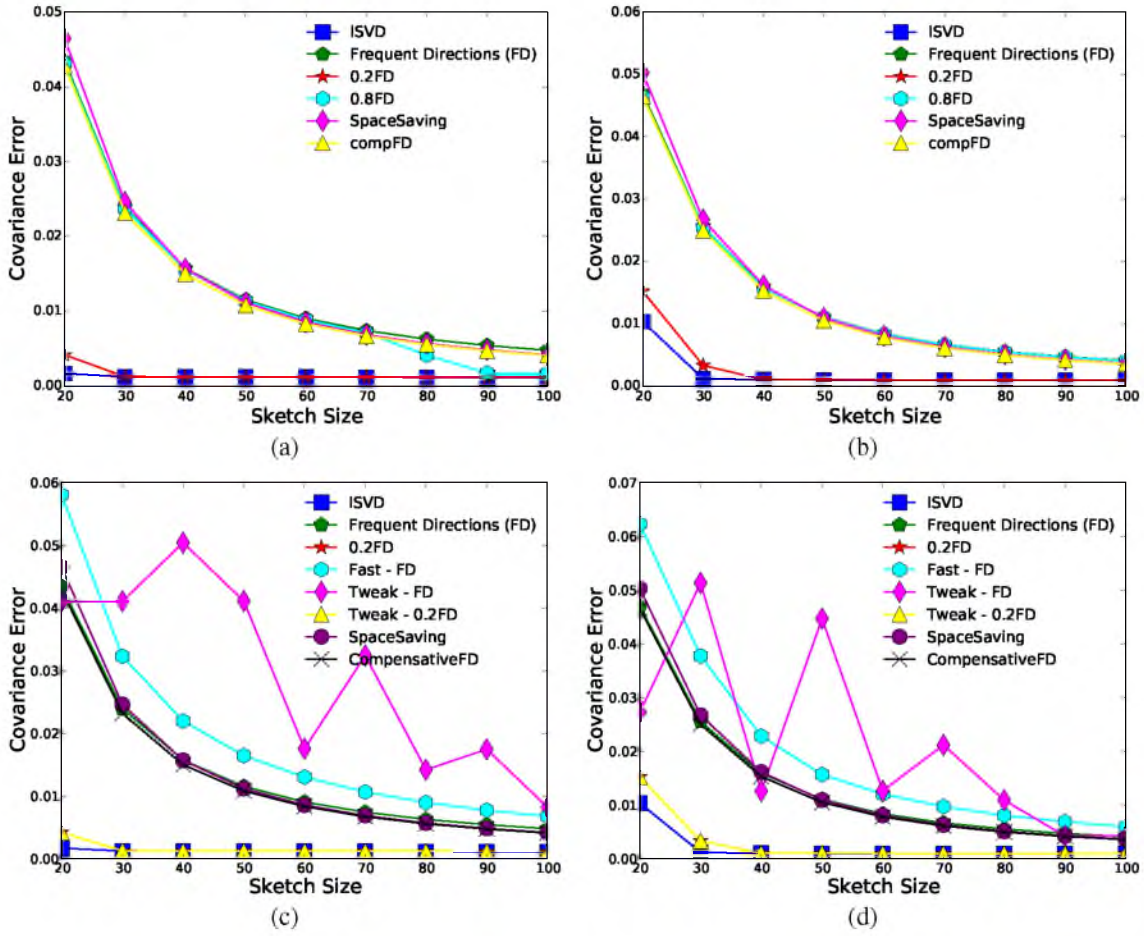

Figure 5.21. SDU $(20, 30)$: Covariance error, Frequent Direction (SDU20), (b) Frequent Direction (SDU30), (c) Frequent Directions with tweaks (SDU20), (d) Frequent Direction with tweaks (SDU30)

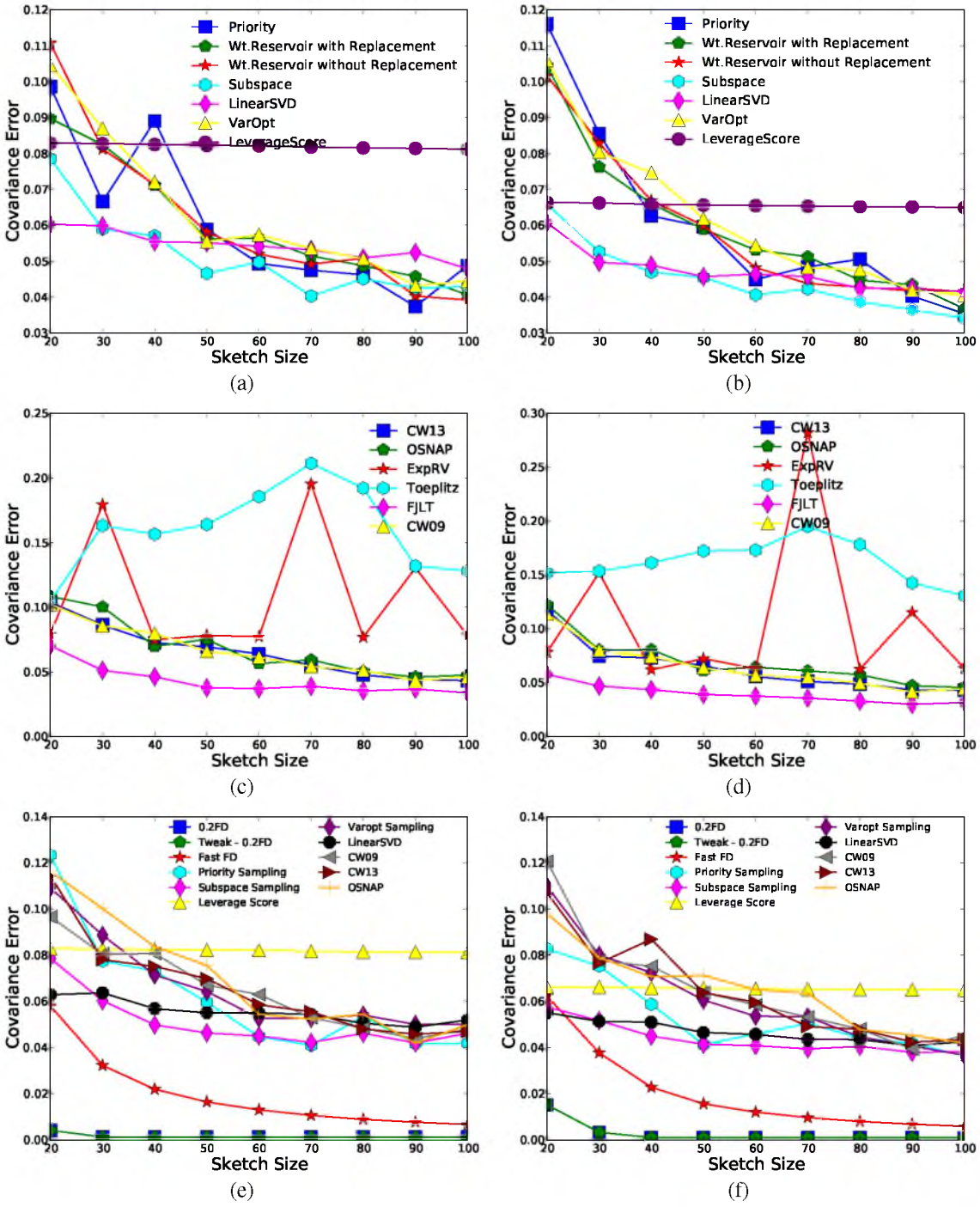

Figure 5.22. SDU $(20, 30)$: Covariance error, (a) Column Sampling (SDU20), (b) Column Sampling (SDU30), (c) Random Projections (SDU20), (d) Random Projections (SDU30), (e) Leading algorithms (SDU20), (f) Leading algorithms (SDU30)

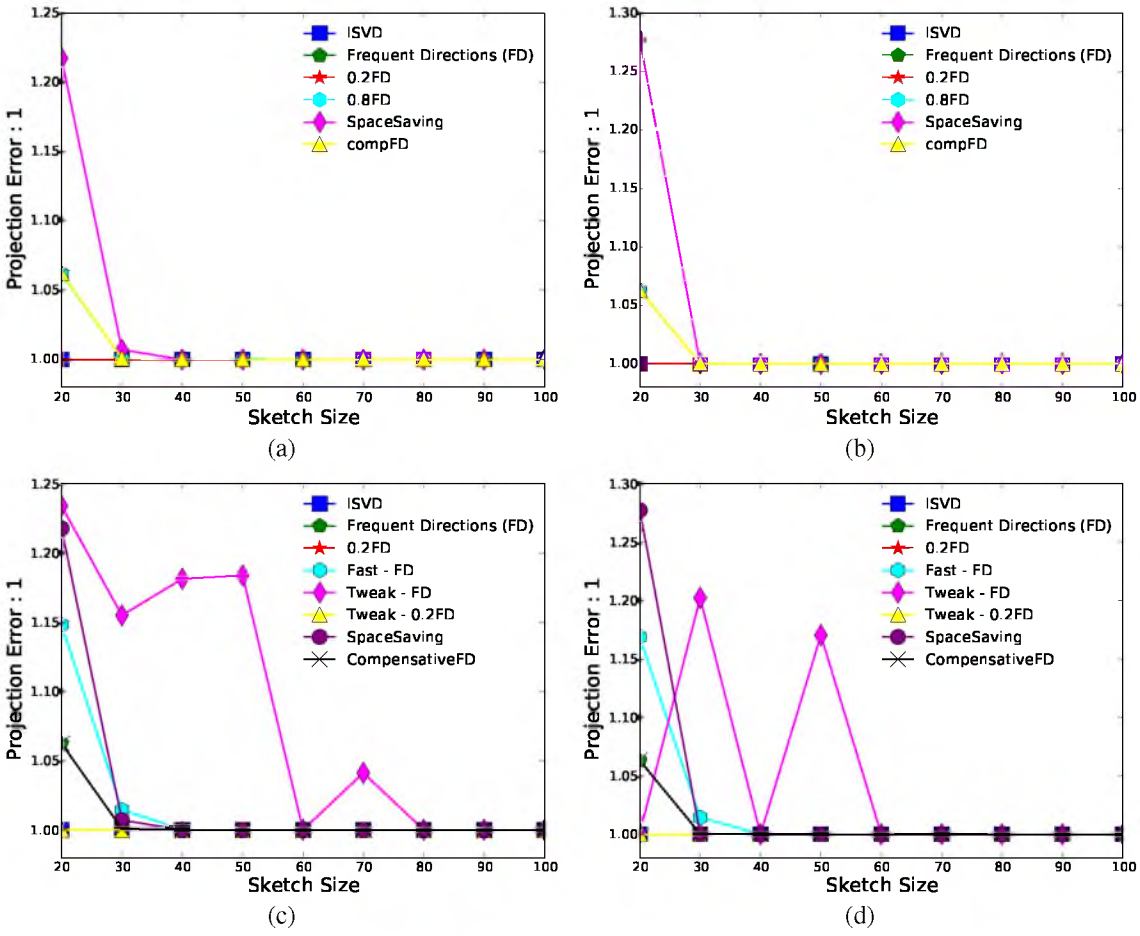

Figure 5.23. SDU $(20, 30)$: Projection error- 1, (a) Frequent Direction (SDU20), (b) Frequent Direction (SDU30), (c) Frequent Directions with tweaks (SDU20), (d) Frequent Direction with tweaks (SDU30)

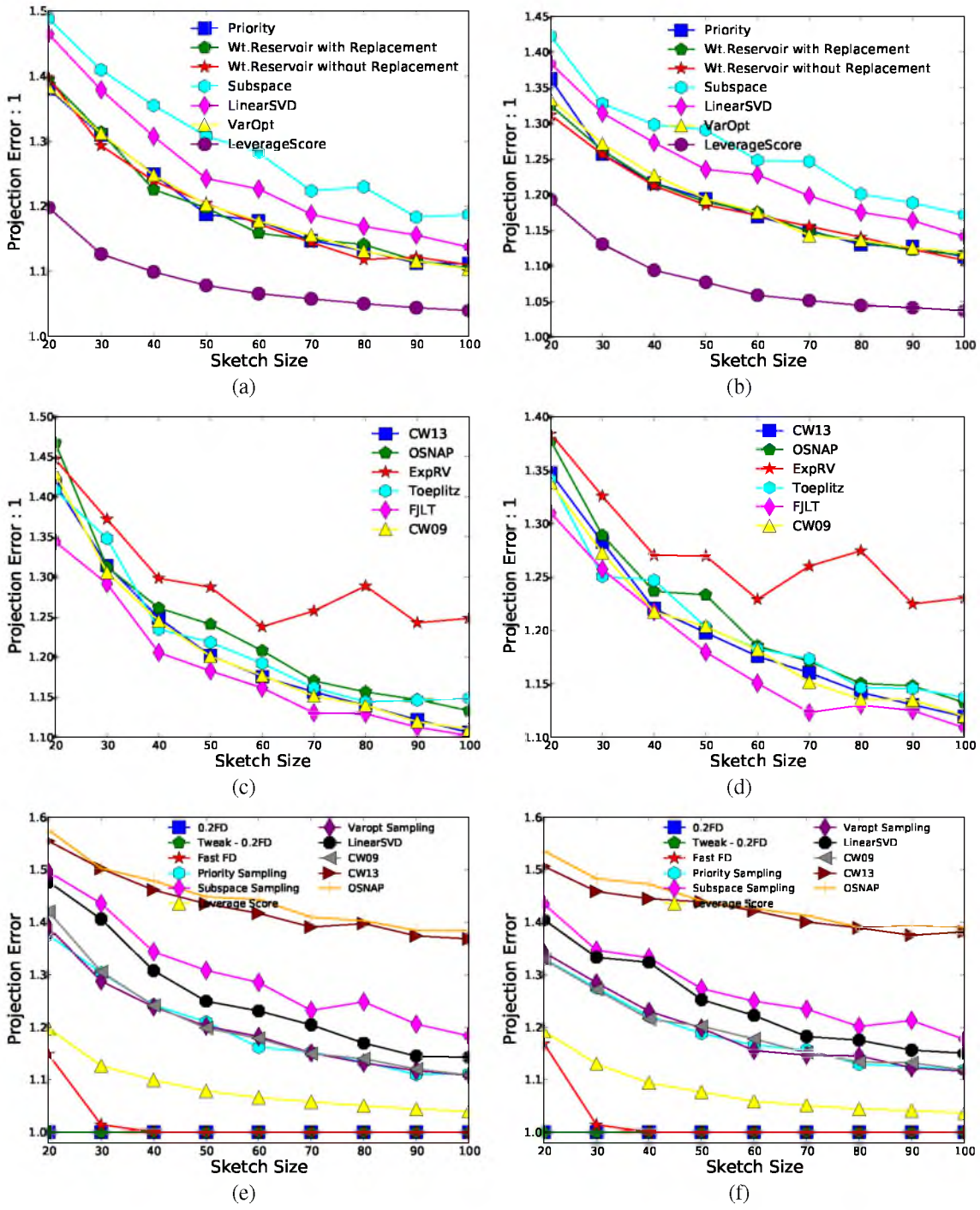

Figure 5.24. SDU $(20, 30)$: Projection error- 1, (a) Column Sampling (SDU20), (b) Column Sampling (SDU30), (c) Random Projections (SDU20), (d) Random Projections (SDU30), (e) Leading algorithms (SDU20), (f) Leading algorithms (SDU30)

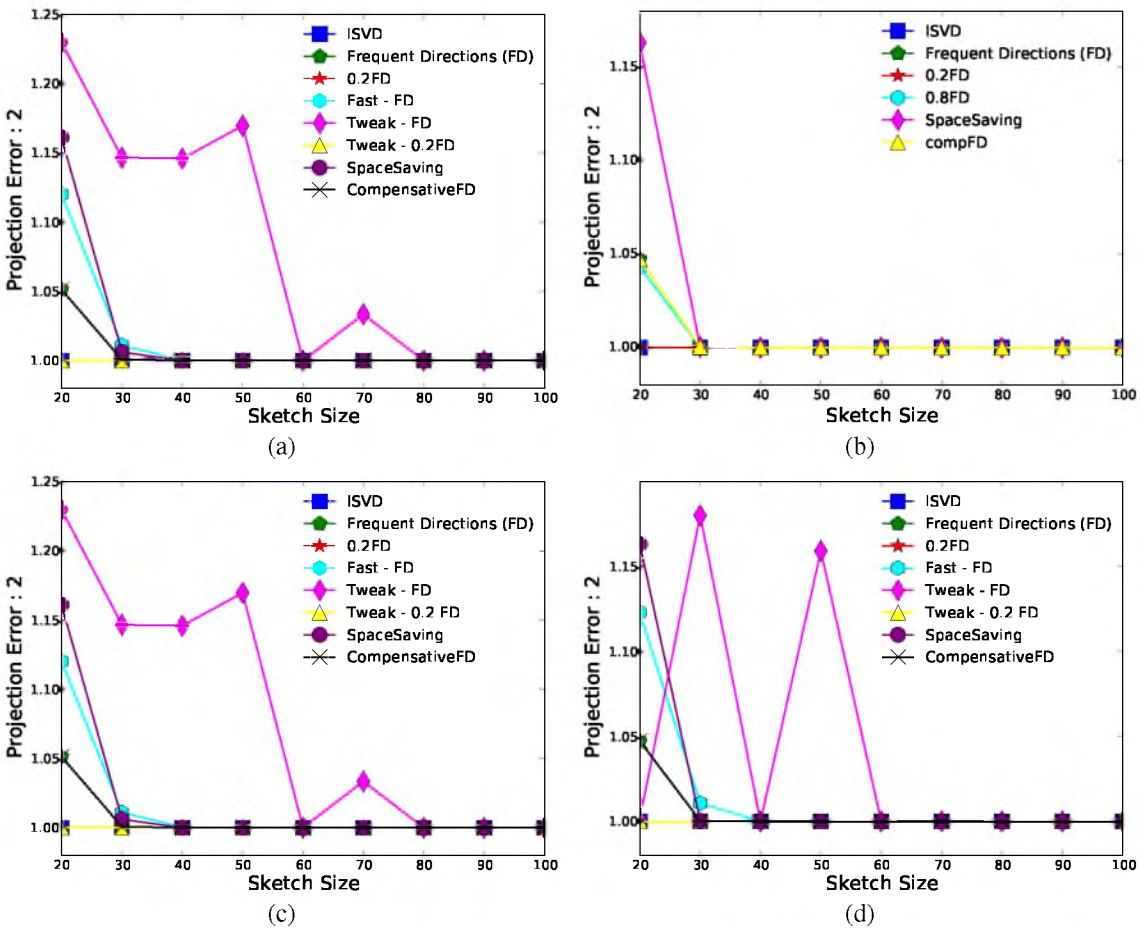

Figure 5.25. SDU $(20, 30)$: Projection error- 2, (a) Frequent Direction (SDU20), (b) Frequent Direction (SDU30), (c) Frequent Directions with tweaks (SDU20), (d) Frequent Direction with tweaks (SDU30)

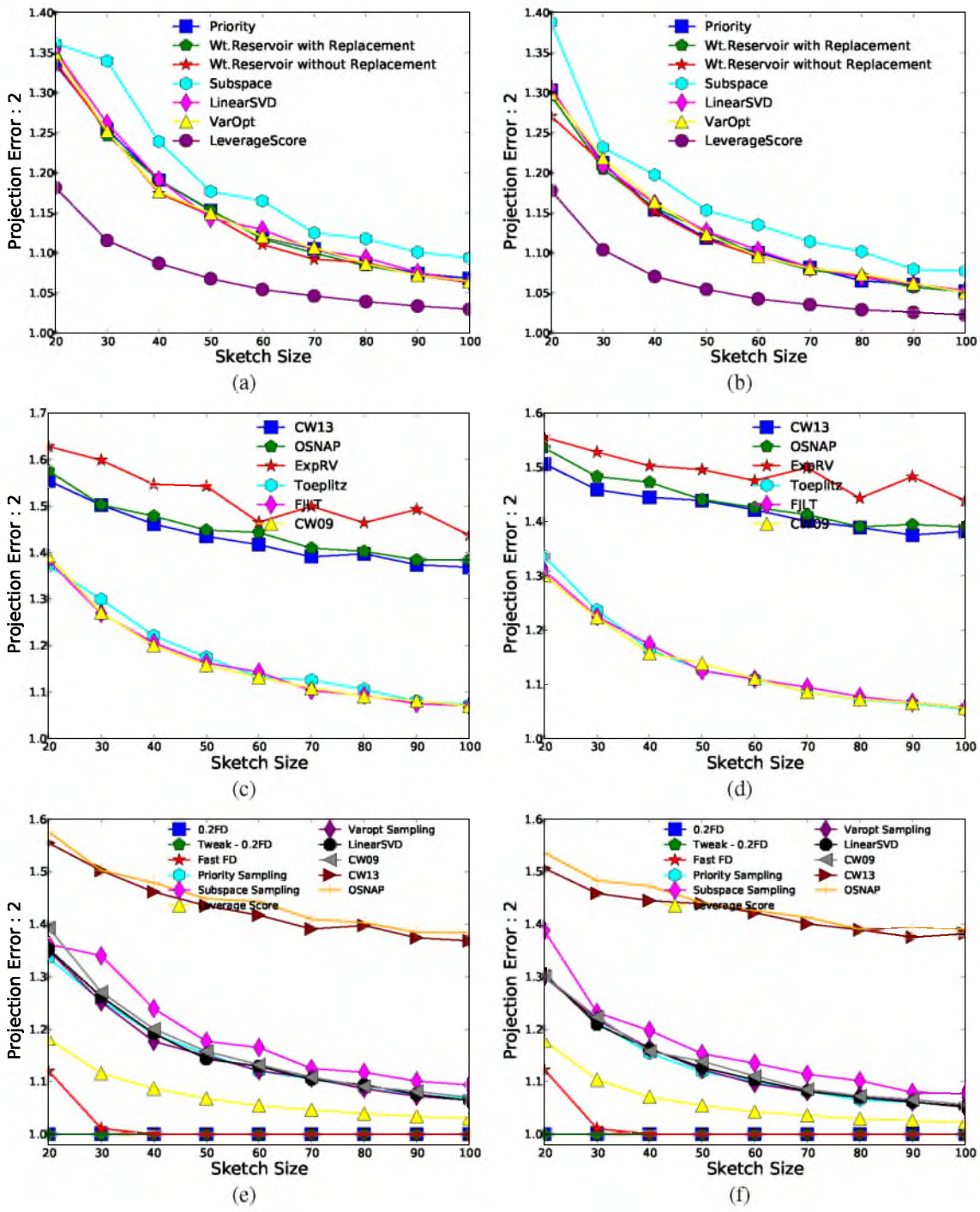

Figure 5.26. SDU $(20, 30)$: Projection error- 2, (a) Column Sampling (SDU20), (b) Column Sampling (SDU30), (c) Random Projections (SDU20), (d) Random Projections (SDU30), (e) Leading algorithms (SDU20), (f) Leading algorithms (SDU30)

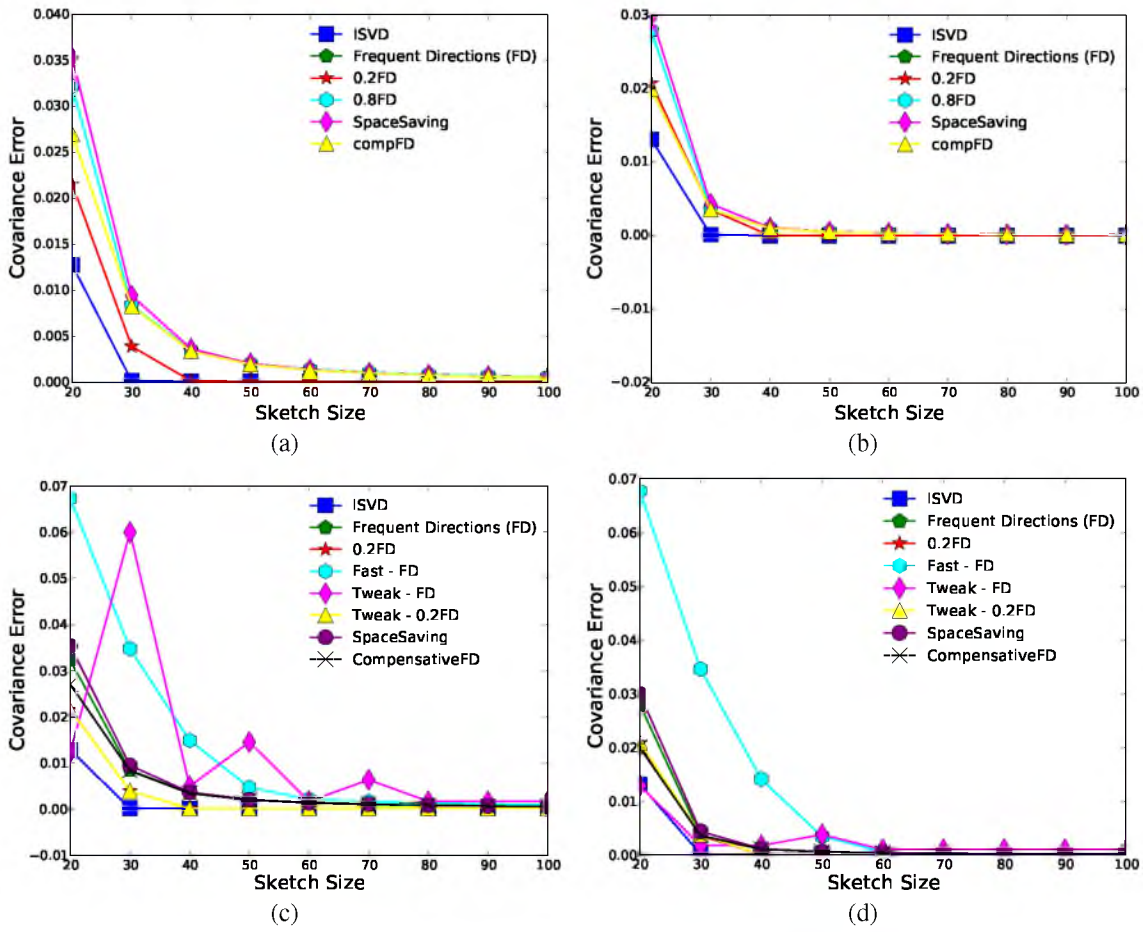

Figure 5.27. SDU $(30, 30), (30, 60)$: Covariance error, (a) Frequent Direction (SDU30_30), (b) Frequent Direction (SDU30_60), (c) Frequent Directions with tweaks (SDU30_30), (d) Frequent Direction with tweaks (SDU30_60)

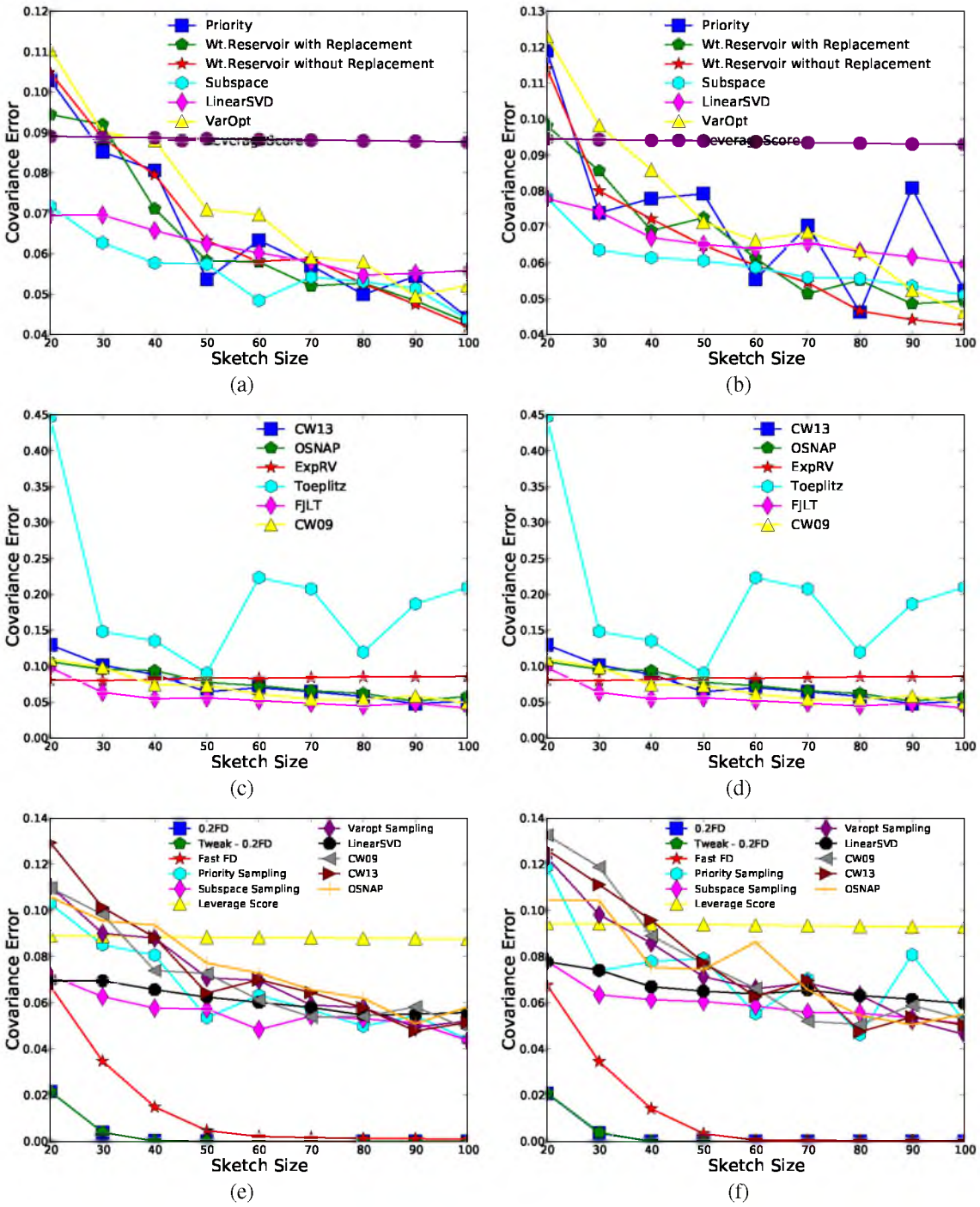

Figure 5.28. SDU $(30, 30)$, $(30, 60)$: Covariance error, (a) Column Sampling (SDU30_30), (b) Column Sampling (SDU30_60), (c) Random Projections (SDU30_30), (d) Random Projections (SDU30_60), (e) Leading algorithms (SDU30_30), (f) Leading algorithms (SDU30_60)

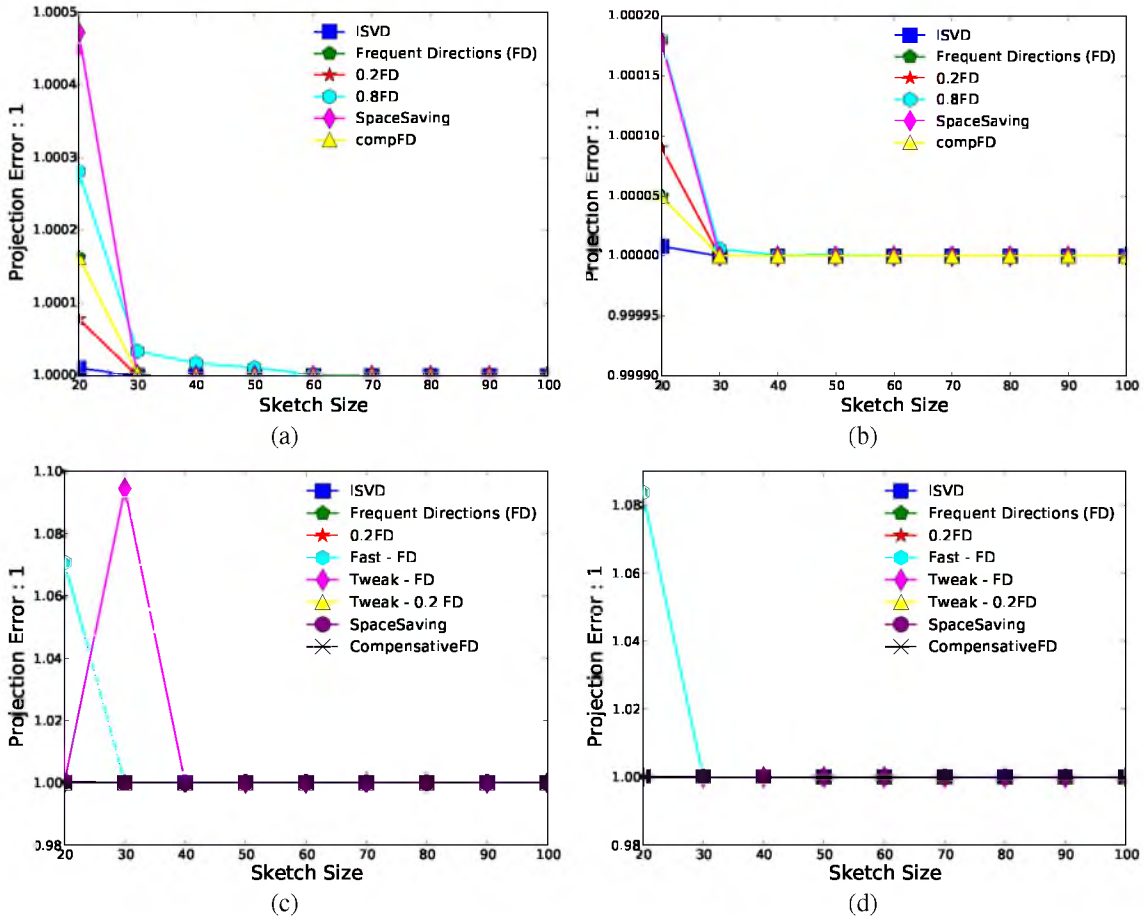

Figure 5.29. SDU $(30, 30), (30, 60)$: Projection error- 1, (a) Frequent Direction (SDU30_30), (b) Frequent Direction (SDU30_60), (c) Frequent Directions with tweaks (SDU30_30), (d) Frequent Direction with tweaks (SDU30_60)

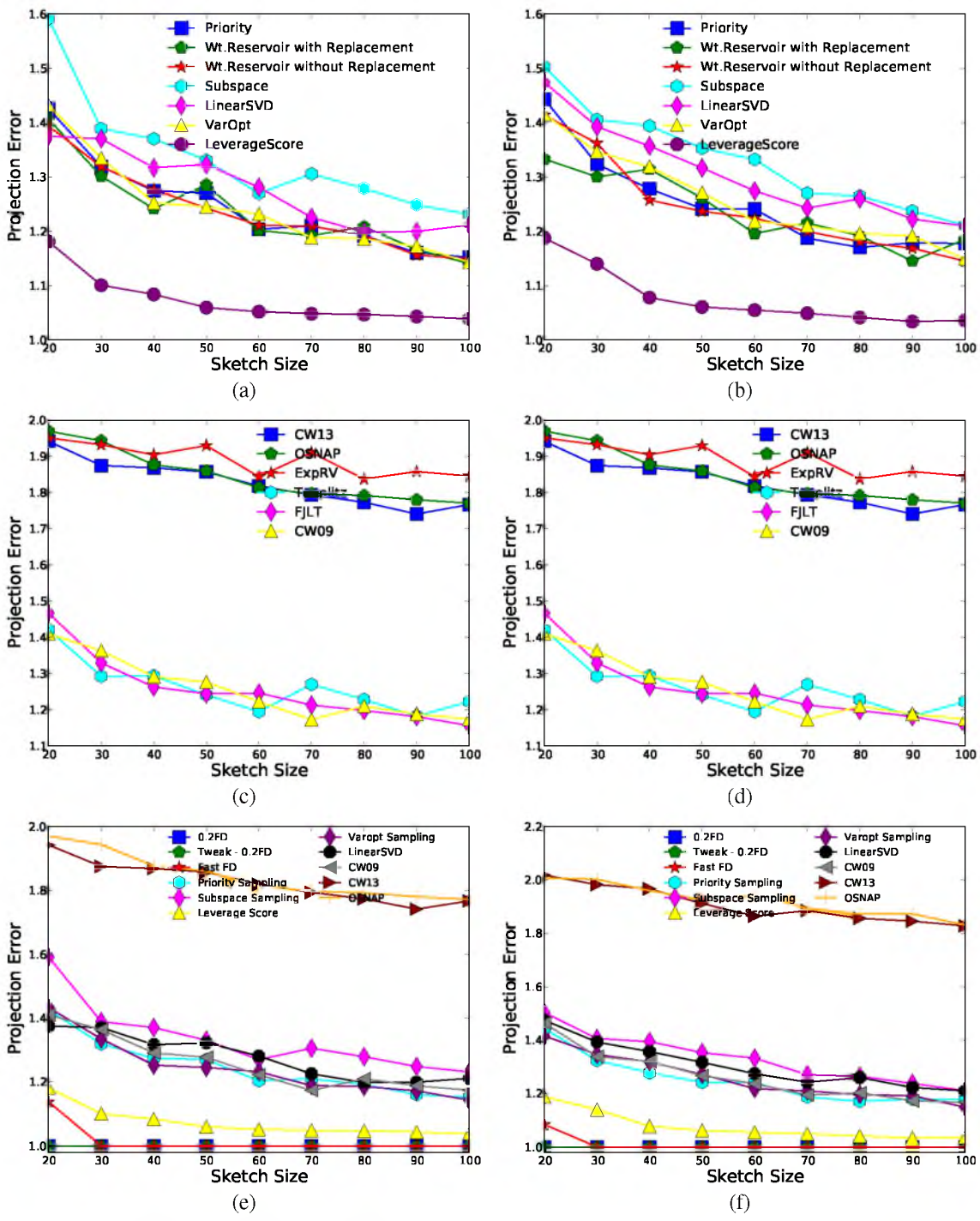

Figure 5.30. SDU $(30, 30), (30, 60)$: Projection error- 1,(a) Column Sampling (SDU30_30), (b) Column Sampling (SDU30_60), (c) Random Projections (SDU30_30), (d) Random Projections (SDU30_60), (e) Leading algorithms (SDU30_30), (f) Leading algorithms (SDU30_60)

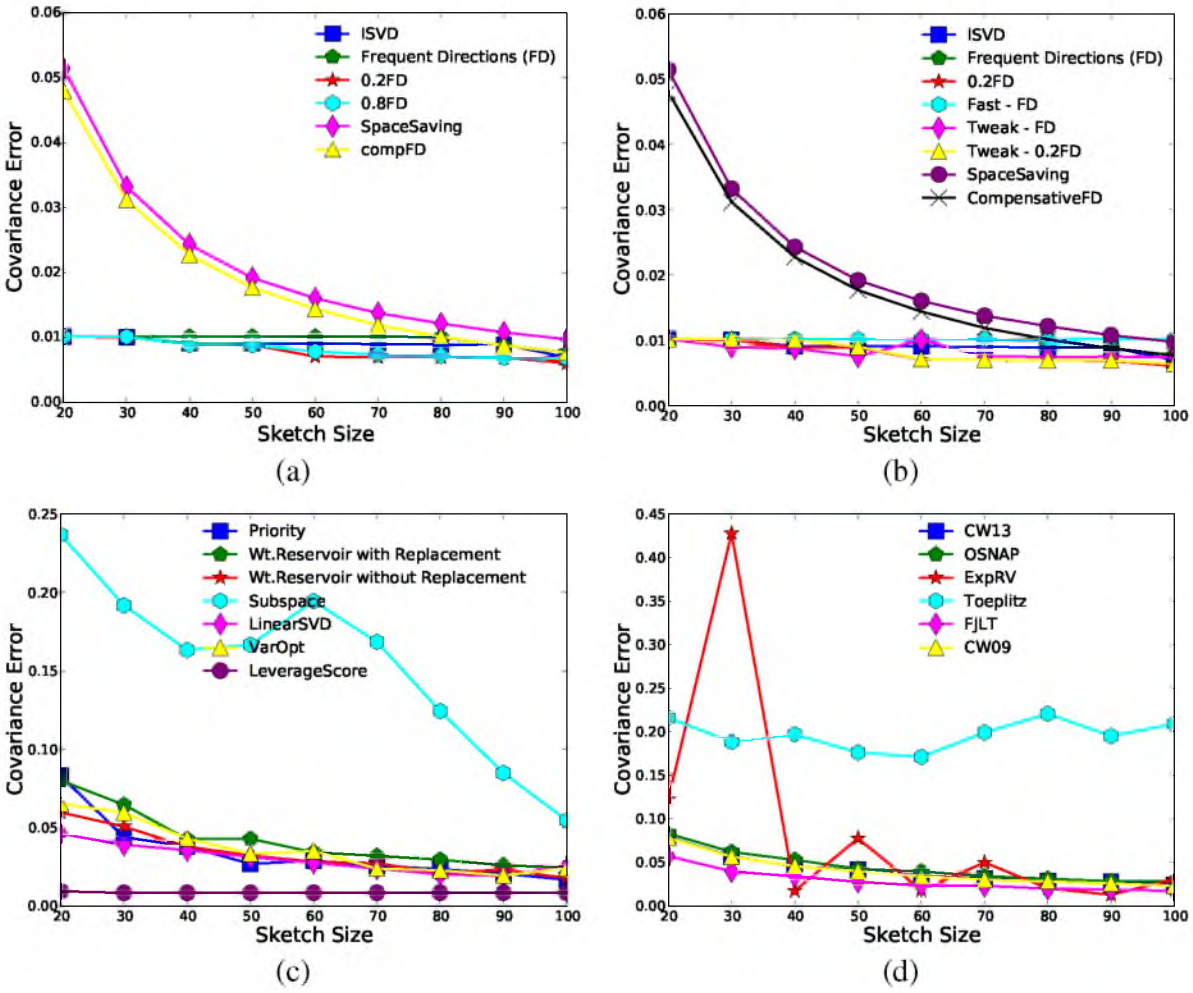

Figure 5.31. QRPivot: Covariance error, (a) Frequent Direction, (b) Frequent Direction with tweaks, (c) Column Sampling, (d) Random Projections

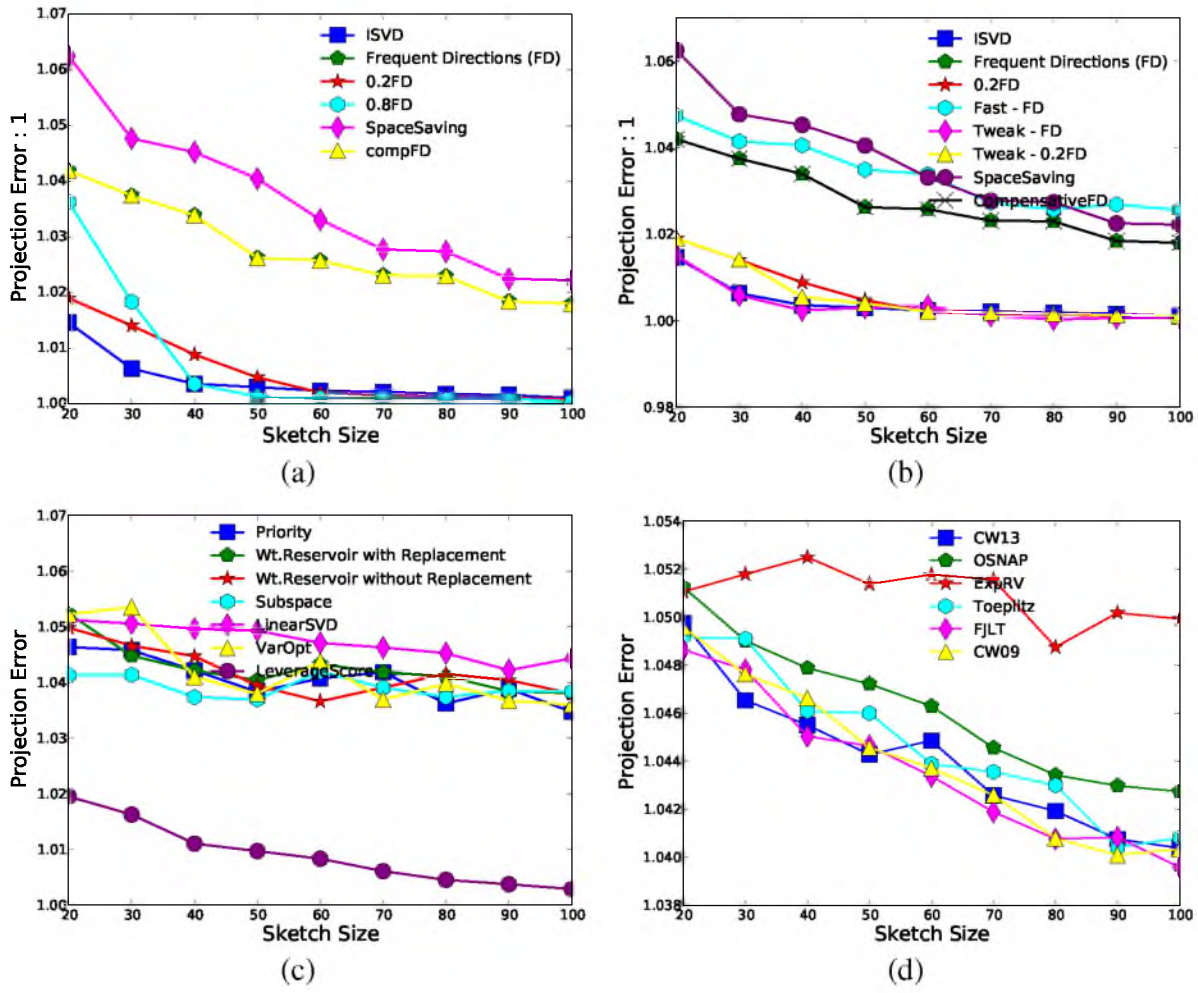

Figure 5.32. QRPivot: Projection error, (a) Frequent Direction, (b) Frequent Direction with tweaks, (c) Column Sampling, (d) Random Projections

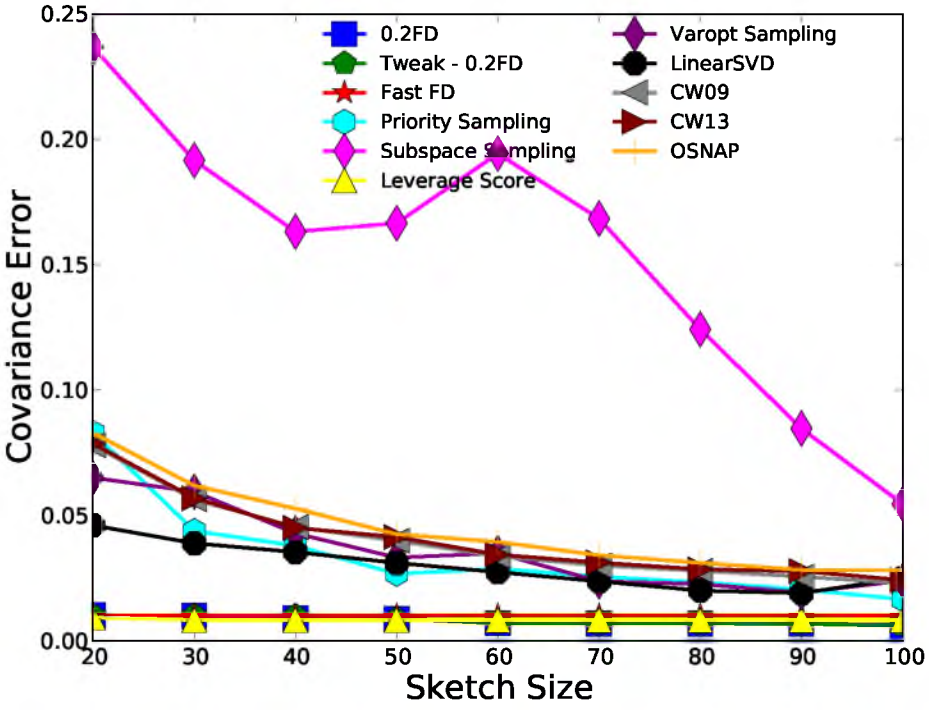

Figure 5.33. Leading algorithms for QRPivot: Covariance error

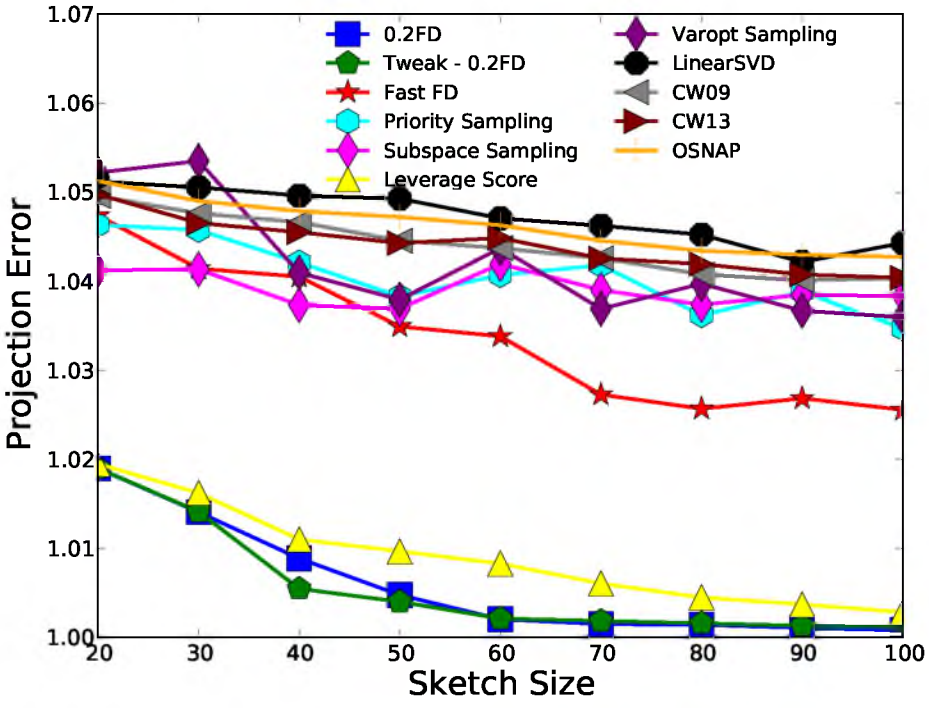

Figure 5.34. Leading algorithms for QRPivot: Projection error

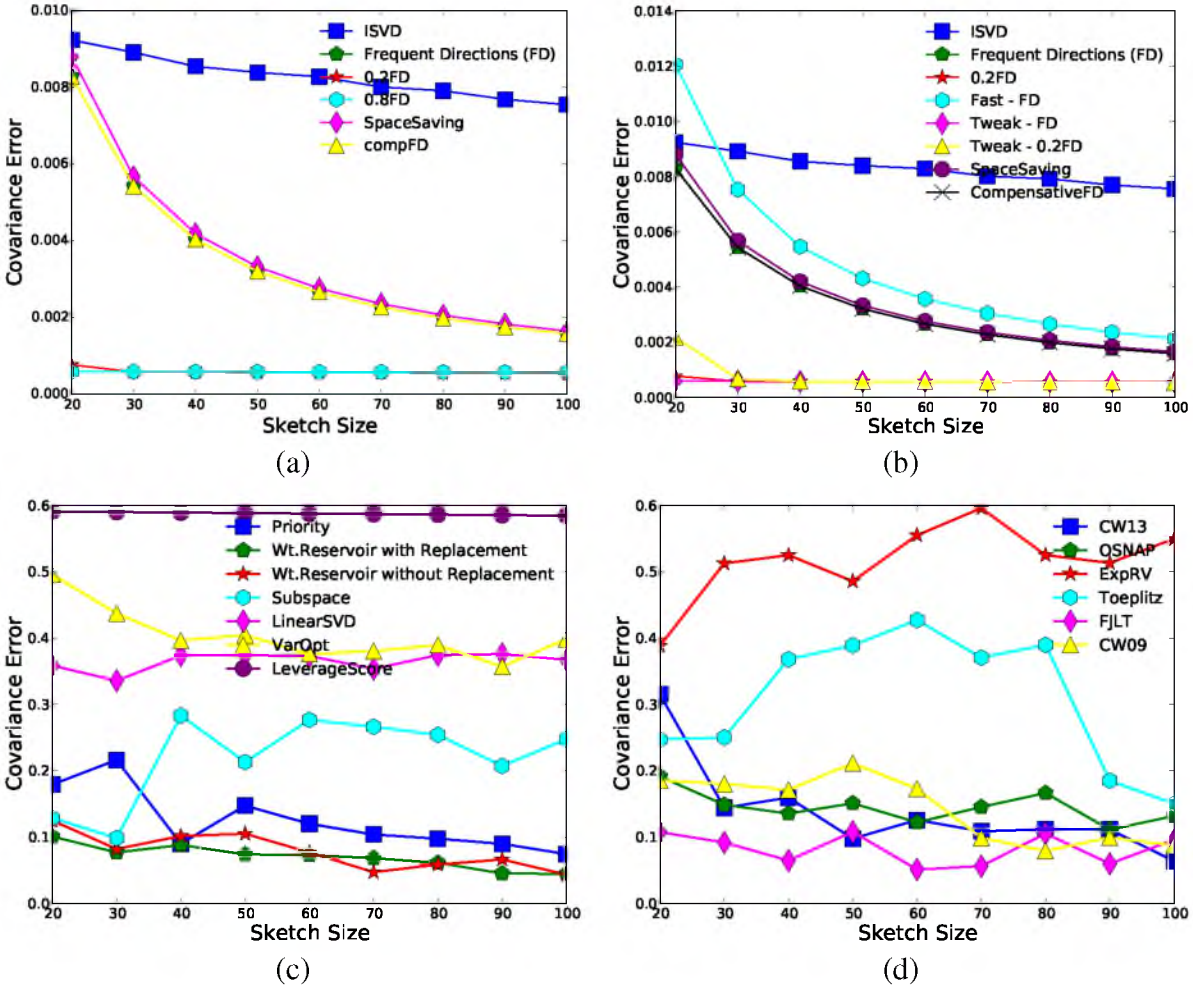

Figure 5.35. Adversarial drift: Covariance error, (a) Frequent Direction, (b) Frequent Direction with tweaks, (c) Column Sampling, (d) Random Projections

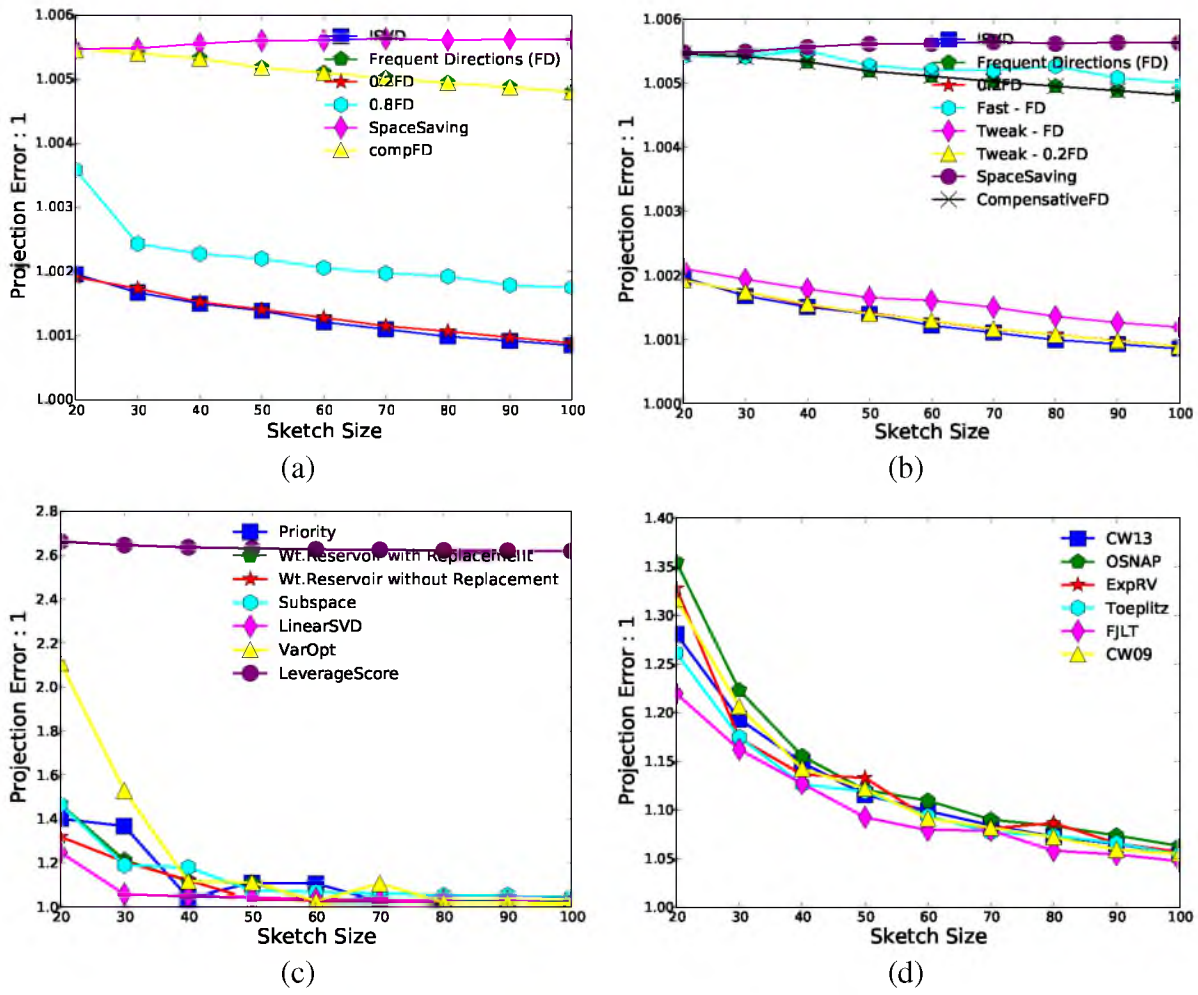

Figure 5.36. Adversarial drift: Projection error, (a) Frequent Direction, (b) Frequent Direction with tweaks, (c) Column Sampling, (d) Random Projections

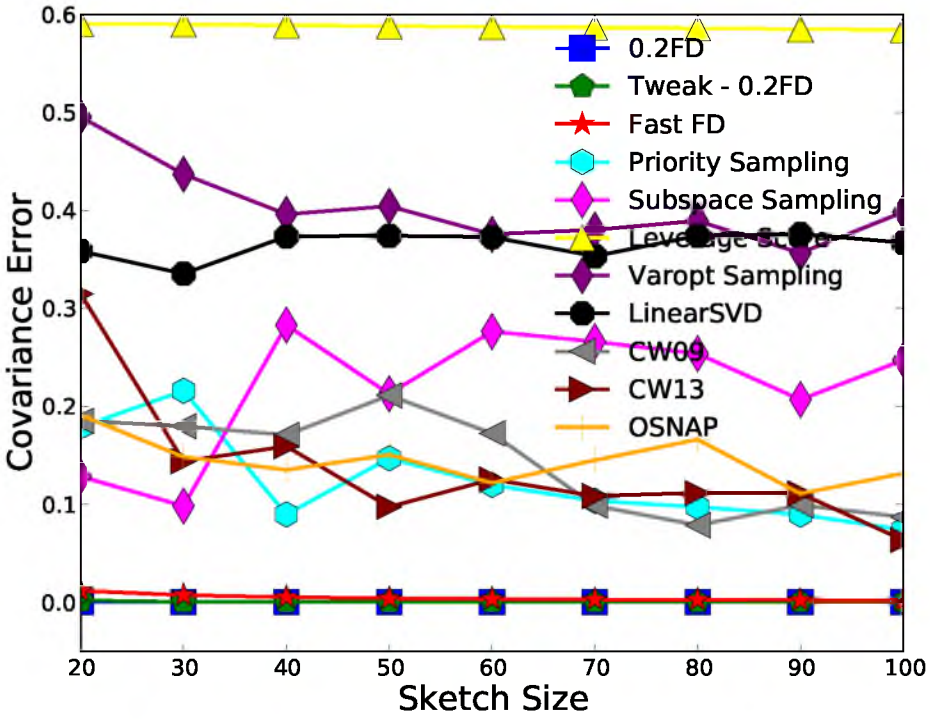

Figure 5.37. Leading algorithms for adversarial drift: Covariance error

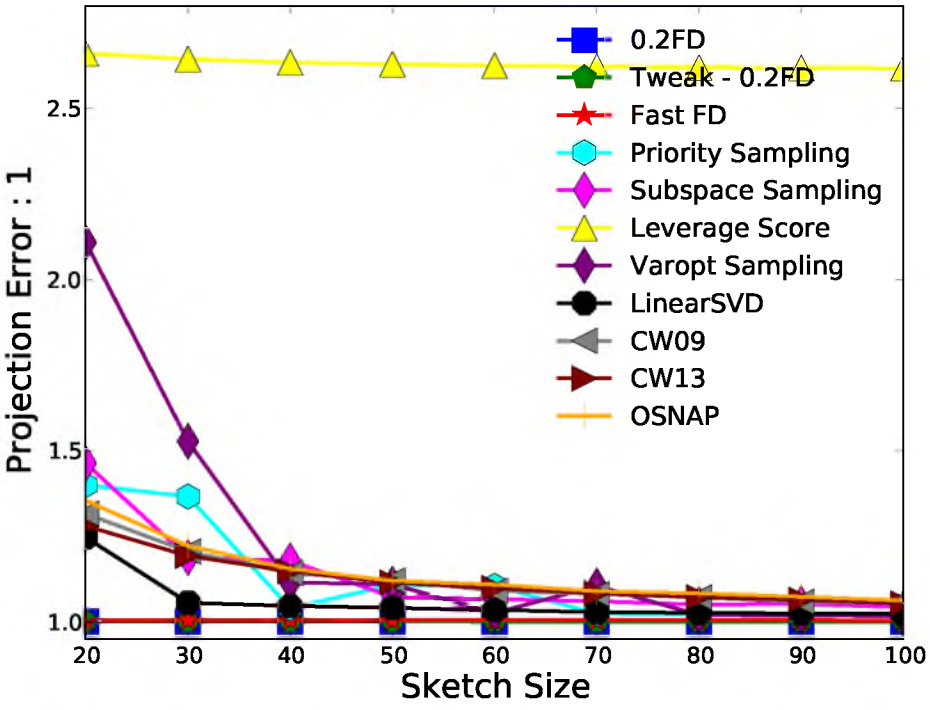

Figure 5.38. Leading algorithms for adversarial drift: Projection error

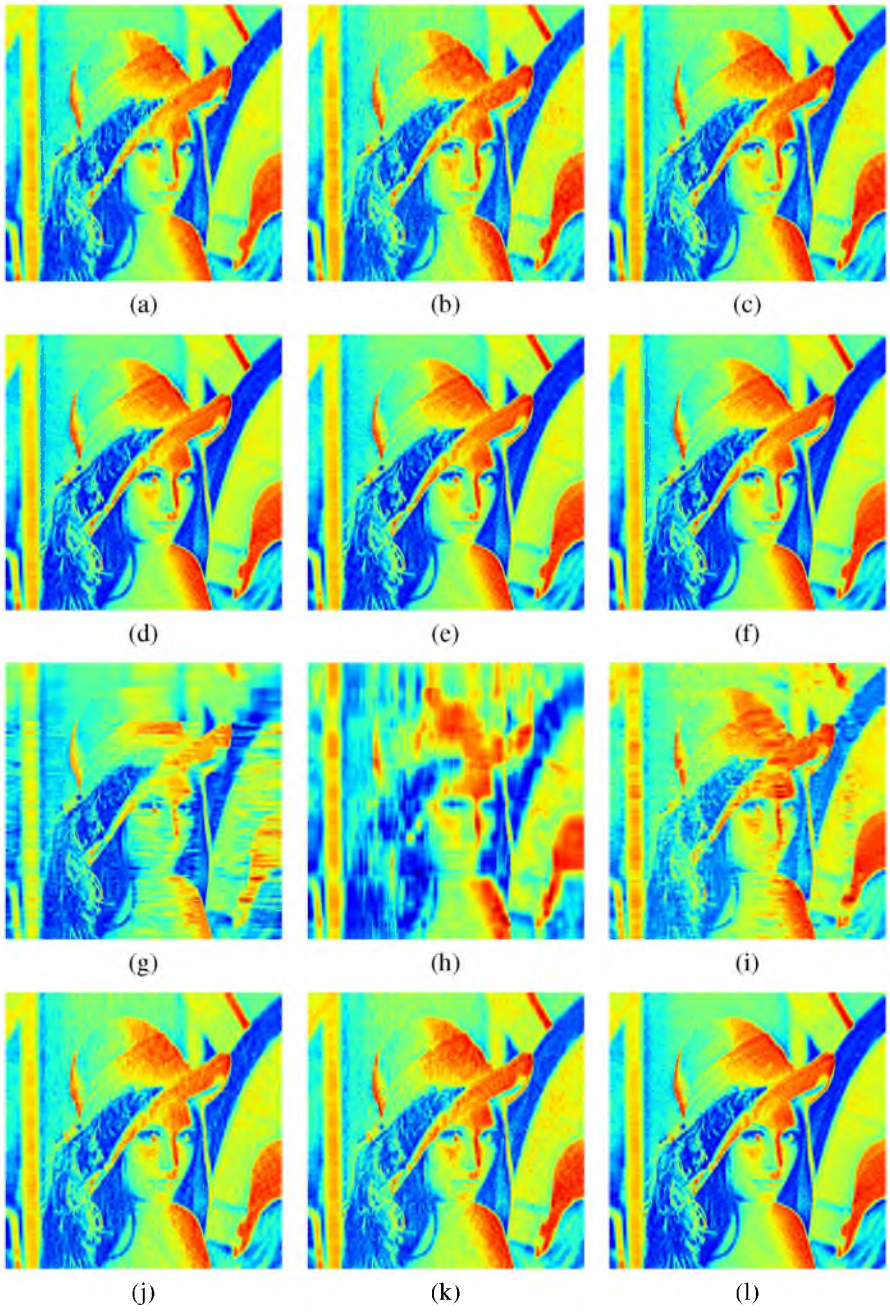

Figure 5.39. Lena: low-rank approximation by leading algorithms, (a) Priority Sampling, (b) Toeplitz, (c) Random Projection, (d) α – FREQUENTDIRECTIONS, (e) ISVD, (f) FREQUENTDIRECTIONS, (g) Det.LeverageScores, (h) LINEAR TIME SVD, (i) SUBSPACE SAMPLING, (j) CW TRANSFORM - 13, (k) OSNAP, (l) Truncated SVD

CHAPTER 6

CONCLUSION

- As LINEAR TIME SVD performs almost as well as RANDOM PROJECTION, can we get relative error guarantees for importance sampling under certain conditions?
- Reservoir sampling algorithms, PRIORITY SAMPLING and VARIANCE OPTIMAL SAMPLING in particular, also match the existing algorithms with theoretical guarantees as does TOEPLITZ; can we get relative error guarantees for reservoir sampling and Toeplitz matrix as a subspace embedding?
- Can we improve the running time of FREQUENTDIRECTIONS and its variants for dense as well as sparse matrices?
- Can we bridge the gap between dense random projections and sparse random projections, which would work for both set of matrices?
- How can we make the sketch matrices interpretable? Row sampling reduces the number of rows, but not dimensions. Usually in machine learning, we do classification, clustering on rows, can we do the same for output by generated FREQUENTDIRECTIONS variants, RANDOM PROJECTION variants, and row sampling algorithms?

In our experiments, FREQUENTDIRECTIONS algorithms give the best accuracy error, but have the worst running time. We demonstrate reservoir sampling algorithms like PRIORITY SAMPLING and VARIANCE OPTIMAL SAMPLING matching its accuracy in much smaller running time, but requiring much larger number of samples/sketch size. We also demonstrate empirically and theoretically Tweak- $0.2-$ FREQUENTDIRECTIONS matching FREQUENTDIRECTIONS for accuracy and being much faster than the other variants while being competitive to Fast - FREQUENTDIRECTIONS algorithm for running time.

REFERENCES

[1] http://www.cise.ufl.edu/research/sparse/matrices/mathworks/qrpivot.html.

[2] http://www.vision.caltech.edu/visipedia/cub-200-2011.html.

[3] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003.

[4] Dimitris Achlioptas and Frank McSherry. On spectral learning of mixtures of distributions. In *Learning Theory*, pages 458–469. Springer, 2005.

[5] Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transfohrm. In *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*, pages 557–563. ACM, 2006.

[6] Yossi Azar, Amos Fiat, Anna Karlin, Frank McSherry, and Jared Saia. Spectral analysis of data. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, pages 619–626. ACM, 2001.

[7] Michael W Berry, Susan T Dumais, and Gavin W O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37(4):573–595, 1995.

[8] Christos Boutsidis, Petros Drineas, and Malik Magdon-Ismail. Near-optimal column-based matrix reconstruction. *SIAM Journal on Computing*, 43(2):687–717, 2014.

[9] Mary E Broadbent, Martin Brown, Kevin Penner, I Ipsen, and R Rehman. Subset selection algorithms: Randomized vs. deterministic. *SIAM Undergraduate Research Online*, 3:50–71, 2010.

[10] Kenneth L Clarkson and David P Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, pages 205–214. ACM, 2009.

[11] Kenneth L Clarkson and David P Woodruff. Low rank approximation and regression in input sparsity time. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, pages 81–90. ACM, 2013.

[12] Edith Cohen, Nick Duffield, Haim Kaplan, Carsten Lund, and Mikkel Thorup. Stream sampling for variance-optimal estimation of subset sums. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1255–1264. Society for Industrial and Applied Mathematics, 2009.

[13] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2003.

[14] Amit Deshpande and Santosh Vempala. Adaptive sampling and fast low-rank matrix approximation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 292–303. Springer, 2006.

[15] Petros Drineas, Alan Frieze, Ravi Kannan, Santosh Vempala, and V Vinay. Clustering large graphs via the singular value decomposition. *Machine Learning*, 56(1-3):9–33, 2004.

[16] Petros Drineas, Ravi Kannan, and Michael W Mahoney. Fast monte carlo algorithms for matrices i: Approximating matrix multiplication. *SIAM Journal on Computing*, 36(1):132–157, 2006.

[17] Petros Drineas, Ravi Kannan, and Michael W Mahoney. Fast monte carlo algorithms for matrices ii: Computing a low-rank approximation to a matrix. *SIAM Journal on Computing*, 36(1):158–183, 2006.

[18] Petros Drineas, Iordanis Kerenidis, and Prabhakar Raghavan. Competitive recommendation systems. In *Proceedings of the Thirty-fourth Annual ACM Symposium on Theory of Computing*, pages 82–90. ACM, 2002.

[19] Petros Drineas, Michael W Mahoney, and S Muthukrishnan. Subspace sampling and relative-error matrix approximation: Column-based methods. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 316–326. Springer, 2006.

[20] Petros Drineas, Michael W Mahoney, and S Muthukrishnan. Relative-error cur matrix decompositions. *SIAM Journal on Matrix Analysis and Applications*, 30(2):844–881, 2008.

[21] Nick Duffield, Carsten Lund, and Mikkel Thorup. Priority sampling for estimation of arbitrary subset sums. *Journal of the ACM (JACM)*, 54(6):32, 2007.

[22] Pavlos S Efraimidis and Paul G Spirakis. Weighted random sampling with a reservoir. *Information Processing Letters*, 97(5):181–185, 2006.

[23] Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. *Journal of the ACM (JACM)*, 51(6):1025–1041, 2004.

[24] Jing Gao, Wei Fan, Jiawei Han, and S Yu Philip. A general framework for mining concept-drifting data streams with skewed distributions. In *SDM*, pages 3–14. SIAM, 2007.

[25] Mina Ghashami, Amey Desai, and Jeff M Phillips. Improved practical matrix sketching with guarantees. In *Algorithms-ESA 2014*, pages 467–479. Springer, 2014.

[26] Mina Ghashami and Jeff M Phillips. Relative errors for deterministic low-rank matrix approximations. *arXiv preprint arXiv:1307.7454*, 2013.

[27] Gene H Golub and Charles F Van Loan. *Matrix Computations*, volume 3. JHU Press, 2012.

[28] Aicke Hinrichs and Jan Vybíral. Johnson-Lindenstrauss lemma for circulant matrices. *Random Structures & Algorithms*, 39(3):391–398, 2011.

[29] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary Mathematics*, 26(189-206):1, 1984.

[30] Ian T Jolliffe. Discarding variables in a principal component analysis. i: Artificial data. *Applied Statistics*, pages 160–173, 1972.

[31] IT Jolliffe. Discarding variables in a principal component analysis. ii: Real data. *Applied Statistics*, pages 21–31, 1973.

[32] Ravindran Kannan, Hadi Salmasian, and Santosh Vempala. The spectral method for general mixture models. In *Learning Theory*, pages 444–457. Springer, 2005.

[33] Ioannis Katakis, Grigorios Tsoumakas, Evangelos Banos, Nick Bassiliades, and Ioannis Vlahavas. An adaptive personalized news dissemination system. *Journal of Intelligent Information Systems*, 32(2):191–212, 2009.

[34] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. Tracking recurring contexts using ensemble classifiers: An application to email filtering. *Knowledge and Information Systems*, 22(3):371–391, 2010.

[35] Jon M Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[36] Edo Liberty. Simple and deterministic matrix sketching. *CoRR*, abs/1206.0594, 2012.

[37] Edo Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 581–588. ACM, 2013.

[38] Jiří Matoušek. On variants of the Johnson–Lindenstrauss lemma. *Random Structures & Algorithms*, 33(2):142–156, 2008.

[39] Frank McSherry. Spectral partitioning of random graphs. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 529–537. IEEE, 2001.

[40] Xiangrui Meng and Michael W Mahoney. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, pages 91–100. ACM, 2013.

[41] Ahmed Metwally, Divyakant Agrawal, and Amr El. Abbadi. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Transactions on Database Systems*, 31(3):1095–1133, 2006.

[42] Jayadev Misra and David Gries. Finding repeated elements. *Science of Computer programming*, 2(2):143–152, 1982.

[43] Jelani Nelson. Johnson–lindenstrauss notes.

[44] Jelani Nelson and Huy L. Nguyen. OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *Proceedings 54th IEEE Syposium on Foundations of Computer Science*, 2013.

[45] Christos H Papadimitriou, Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. Latent semantic indexing: A probabilistic analysis. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 159–168. ACM, 1998.

[46] Dimitris Papapailiopoulos, Anastasios Kyrillidis, and Christos Boutsidis. Provable deterministic leverage score sampling. *arXiv preprint arXiv:1404.1530*, 2014.

[47] Prabhakar Raghavan and Monika R Henzinger. Computing on data streams. In *External Memory Algorithms: DIMACS Workshop External Memory and Visualization, May 20-22, 1998*, volume 50, page 107. American Mathematical Soc., 1999.

[48] Robert Ricci, Gary Wong, Leigh Stoller, Kirk Webb, Jonathon Duerig, Keith Downie, and Mike Hibler. Apt: A platform for repeatable research in computer science. *ACM SIGOPS Operating Systems Review*, 49(1), January 2015.

[49] Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 143–152. IEEE, 2006.

[50] Mario Szegedy. The dlt priority sampling is essentially optimal. In *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*, pages 150–158. ACM, 2006.

[51] Haixun Wang, Wei Fan, Philip S Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–235. ACM, 2003.

[52] David Woodruff and Qin Zhang. Subspace embeddings and ℓ_p-regression using exponential random variables. In *Conference on Learning Theory*, pages 546–567, 2013.